Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

1989

Design and impementation of a debugger for MC68020 based Educational Computer Board.

Uzunsokakli, Mustafa Yavuz

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/25821

Downloaded from NPS Archive: Calhoun

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

DUD
NAVA
13-6002

NAVAL POSTGRADUATE SCHOOL

Monterey, California

U96

THESIS

DESIGN AND IMPLEMENTATION OF A DEBUGGER FOR MC68020 BASED EDUCATIONAL COMPUTER BOARD

by

Mustafa Yavuz Uzunsokakli

December, 1989

Thesis Advisor: Gerald J. Lipovski

Approved for public release; distribution is unlimited.

T248099

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) 62	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a NAME OF FUNDING / SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO.

11 TITLE (Include Security Classification)
DESIGN AND IMPLEMENTATION OF A DEBUGGER FOR MC68020 BASED EDUCATION COMPUTER BOARD

12 PERSONAL AUTHOR(S)
UZUNSOKAKLI, Mustafa Yavuz

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1989 December	180

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official position or policy of the Department of Defense or the US Government.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Microprocessor Development System; Debugger

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

A debugger has been designed and imlemented to debug MC68020 assembly language programs which run on an MC68020-based Educational Computer Board (ECB). The debugger consists of two physically separate modules and runs on both a Macintosh and on the ECB. The debugger and the ECB communicate via an RS232 interface at a Baud rate of 9600.

In addition to basic debugger commands for the MC68020, the debugger also supports commands which enable the user to examine or modify the MC68881 Coprocessor's registers. An important feature is that it is user-friendly. It utilizes pull-down menus, where the user can select and execute the desired command simply by clicking the mouse. This debugger and a LightspeedC compiler provides the user with an integrated environment, where he or she can edit, assemble and debug assembly language programs.

Applications of this software tool, and the accompanying ECB, can be used for both research and teaching. For example, it can replace the current system that supports the Naval Postgraduate School course EC2800.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL BUTLER, Jon T.	22b TELEPHONE (Include Area Code) 408-646-3299 / 22c OFFICE SYMBOL 62Bu

DD Form 1473, JUN 86 *Previous editions are obsolete.*
S/N 0102-LF-014-6603

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Approved for public release; distribution is unlimited.

Design and Implementation of a Debugger

for

MC68020 Based Educational Computer Board

by

Mustafa Yavuz Uzunsokakli
Lieutenant Junior Grade, Turkish Navy
B.S.E.E., Turkish Naval Academy, 1983

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1989

ABSTRACT

A debugger has been designed and implemented to debug MC68020 assembly language programs which run on an MC68020-based Educational Computer Board (ECB). The debugger consists of two physically separate modules and runs on both a Macintosh and on the ECB. The debugger and the ECB communicate via an RS232 interface at a Baud rate of 9600.

In addition to basic debugger commands for the MC68020, the debugger also supports commands which enable the user to examine or modify the MC68881 Coprocessor's registers. An important feature is that it is user-friendly. It utilizes pull-down menus, where the user can select and execute the desired command simply by clicking the mouse. This debugger and a LightspeedC compiler provides the user with an integrated environment, where he or she can edit, assemble and debug assembly language programs.

Applications of this software tool, and the accompanying ECB, can be used for both research and teaching. For example, it can replace the current system that supports the Naval Postgraduate School course EC2800.

Thesis
U96
C.1

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

There has been a very rapid growth in the use of microprocessors. With the advent of physically smaller but functionally more capable and faster microprocessors, microprocessor systems, besides being cheap and practical, are becoming almost equivalent to the capability and speed of main-frame computers of the past. Considering all these reasons, a complete and thorough understanding of the capabilities of microprocessors and microcomputers is essential. A microprocessor software development system is a necessary part of this.

The debugger created in this thesis study is software part of a complete MC68020 microprocessor development system. The hardware part is the MC68020 based Educational Computer Board (ECB), which is developed by Tugcu[1989]. In fact, the development of the software and the hardware was done simultaneously.

A debugger for the MC68000 (called Tutor Monitor [Ref. 1]), was created by the Motorola Company for training and operational use. As far as the execution of a user program is concerned, this debugger is capable of doing the same things which the Tutor Monitor can do. To be more accurate, the collection of commands provided in this debugger is a proper subset of the commands present in the Motorola's Tutor Monitor. Since the announcement of the Motorola's ECB, ten years ago, there have been significant improvements in microprocessor speed, instruction sets, etc. Also, the utilization of coprocessors has added more precision in scientific calculations. This debugger, which was designed to be used in debugging MC68000-MC68020 assembly language programs (for MC68020 instructions, see Ref. 2), is also capable of handling MC68881 Coprocessor-related instructions (for MC68881 instructions, see Ref. 3),

thereby giving the user higher precision debug capabilities. This feature is not present in Tutor Monitor.

Part of this debugger, the monitor, runs on the ECB and carries out the commands sent by the main program. The monitor program also includes communication routines. With the use of these two routines, serial communication is done in software.

One of the advantages of this system is that it does not require an extra dumb terminal, which is needed for the Motorola's ECB System. Thus, this debugger can be viewed as an up-to-date version of the Motorola's debugger.

II. A GENERAL OVERVIEW OF DEBUGGERS AND ASSEMBLY LANGUAGE

The goal of this chapter is to give a general background about assembly language and debuggers. Assuming that this debugger is used as a teaching aid in a microprocessor architecture course (e.g., EC2800 at the Naval Postgraduate School) and the student is a beginner in this area, the information contained in this chapter will serve as introduction.

A. WHAT IS AN ASSEMBLY LANGUAGE ?

An assembly language is a level of language between the machine language and high level language. Machine language consists of a series of binary digits which is the computer can interpret directly but is very hard for humans to use. In assembly language, each machine instruction is represented by a mnemonic, and there are no binary digits. For example, it is a lot easier to remember the words like MOVE, ADD, SUB, etc., than to remember a series of binary digits corresponding to these instructions. Essentially, assembly language is an English-like version of machine language, and there is a one-to-one correspondence between instructions in these two languages.

In addition to representing the instructions by mnemonics, memory locations can also be given labels. In this way the assembler keeps track of the addresses rather than the programmer.

With the design of the intelligent compilers, high level languages became more capable and more widely used. Meanwhile, assembly language became less practical and less important in comparison. High level languages are easier to learn and most

importantly, they are portable. On the other hand, assembly languages are not that simple and portable. They are heavily machine dependent.

Despite the disadvantages mentioned above, assembly language is still used and has some advantages over high level languages. Assembly language presents all the available resources of the processor to the user. It allows more effective code (sometimes using less memory).

1. Format of Assembly Language Programs

Assembly language program statements can be considered to have four parts.

- Label field
- Opcode field
- Operand field
- Comment field

As mentioned above, labels are used to refer to memory locations, as symbols rather than absolute addresses. Labels, usually start in the first column. Depending on the assembler, most labels are followed by a ":".

The opcode field contains the mnemonic for the instruction to be executed. Also, assembler directives such as DC (Define Constant), DB (Define Byte), etc., can be included in this field.

The operand field contains the source and destination locations which will take part in the execution of that instruction. They can be registers or memory locations.

The comment field serves as a place where the programmer can explain his program. Comments are especially helpful in assembly language programs, since such programs are substantially more difficult to understand than high level languages. Without comments, it may not even be possible to understand another programmer's

assembly language program. For these reasons, an assembly language programmer should have the habit of writing down comments.

Two sample assembly language programs are given in Appendix G. Anything preceded by a semicolon is considered to be a comment, which is ignored by the assembler. Sample program #1, copies the elements of an array of bytes A[5] to an array B[5]. When this program is assembled, a listing file is obtained, which is given in Appendix G. There are two more fields in the listing file. These fields are introduced by the assembler. The first shows the addresses and the second field shows the hexadecimal representation of the machine code corresponding to the instructions. By looking at the address field, the user can easily figure out how many bytes of code is produced by each mnemonic instruction. Sample program #2 serves as an example of coprocessor instructions.

B. WHAT IS A DEBUGGER ?

Debuggers are software tools which help in developing and testing programs. These programs might be written in assembly language or in a high level language.

The debugger, created in this thesis study, is designed and implemented to debug MC68020 assembly language programs. By using this debugger, the user can create his assembly language program, assemble it, download it to ECB, and run it. He can also disassemble his code (Disassemble means the hexadecimal representation of memory contents are converted into corresponding mnemonic instructions), display or modify the memory or register contents. In short, he can control the execution of his program.

As an example, let us take the sample program #1 (see Appendix G) and further let us assume that the user sets a **Breakpoint** at address $001A (Dollar sign indicates a hexadecimal number) which is the label LOOP. The user also sets the program counter to $000A which is the beginning address of his program. When he starts to run his

program, each time the breakpoint is reached, execution will stop and control is given to the debugger. He will be able to see the various register and memory contents. At this step he can make some memory or register modifications or he can continue without any change. On the other hand, if the user chooses to **Trace Branch** he will be able to see the same kind of information as many times as a branch is indeed taken.

Another choice may be setting a breakpoint at address $0024 which is the end of user program. If the user does not select any **Trace** option, he will see the information only once, at the end of execution of his program, skipping the intermediate parts. With the selection of a Trace option or by setting breakpoints and breakcounts or just by removing the present breakpoints, the user can have a variety of levels of control when executing his program.

More information about the usage and capabilities of this debugger can be obtained from Appendix F.

III. DESIGN AND IMPLEMENTATION OF THE DEBUGGER

This chapter gives a brief description about the design considerations and implementation of this debugger. More detailed information can be obtained from the appendices which are referenced, when necessary, throughout the chapter.

A. DESIGN OBJECTIVES

The design goals are listed below:

- This debugger should be user-friendly.
- It should be capable of supporting essential debugger commands.
- It should also support MC68881 Coprocessor related commands.

An important aspect about user-friendliness is that the user should not be forced to memorize a number of commands. For this reason, it was decided to implement this debugger on a Macintosh computer. The pull-down menu capabilities of the Macintosh made it possible for this debugger to be a menu-driven software tool.

On the other hand, only basic debugging commands were supported for reasons of simplicity. These commands are the most widely used. In general, these commands can be put in three categories:

1. Memory display/modify commands
2. Register (either microprocessor's or its coprocessor's) display/modify commands
3. Control commands (e.g., setting a breakpoint, tracing, etc.)

After determining the design objectives, it was further decided to design the debugger as two separate modules. This was an inevitable result of the fact that this

debugger is to be used in debugging assembly language programs which run on the ECB. As a result, the following two modules are implemented:

- The main program which is mostly written in C and runs on a Macintosh. (Part of main program, which does the disassembly, is written in assembly language.)
- The support program (monitor) which is written in assembly language and runs on the ECB.

The distribution of the functions to the main program and the monitor program are described in the following two subsections.

1. The Functions of the Main Program

The functions provided by the main program are listed as follows:

- Display menus through which the user interface is provided.
- Alert the user if something goes wrong either in the Macintosh or in the ECB.
- Provide serial communication with the ECB, and with the printer at a Baud rate of 9600.

2. The Functions of the Monitor Program

The functions provided by monitor program can be listed as follows:

- Provide serial communication with the Macintosh at a Baud rate of 9600.
- Execute the command which is sent by the main program.
- Provide an **Abort** option to the user.

In the following two sections, a brief discussion is given of the design and implementation of these two modules.

B. MAIN PROGRAM

The main program consists of functions contained in five different files. They are monitor.c, download.c, disasm.c, menu.c, and test.c. Except for the code written for the disassembler which is written in assembly language, the rest of the code is written in C Language. The source code for the disassembler in the Tutor Monitor [Ref. 1] is obtained from the Motorola Company, and then with the addition of some changes, it was adapted to the Macintosh.

When the debugger is first run, it starts execution with an initialization step. The support of serial communication with ECB and serial printer, through the modem port, and serial printer port respectively, are done in download.c during the initialization process. The allocation of the required input and output buffers, the baud rate, etc., are all done on the entry to the program main().

In order to get the **start** and **end** addresses of the user program which is to be downloaded, test.c is run once. After that, main() displays the main menu, awaiting a user command which could be the execution of a single function such as Clearscreen or the selection of any particular menu.

The choice of menus are listed below:

1. Options menu
2. Registers menu
3. Floating Point Registers menu
4. Memory display menu
5. Memory modify menu
6. Go menu.

The user can do different things in different menus. A flowchart for each menu is given in Appendix A. This gives a clear understanding about how the menus are

organized and how switching occurs between the menus. In addition to the information given in Appendix A, more information, such as creating the menus, implementing the user interface, etc., is given in Appendix C, where all the source code of the debugger are shown.

C. MONITOR PROGRAM

This is a support program for the main debugger which runs on the Macintosh. The monitor program is EPROM resident and runs both in RAM and ROM. While running, it occupies lower RAM address space. The addresses below 1000 hexadecimal are reserved for the monitor program. The user program should not reside in the memory locations which are reserved for the system. Even though the debugger runs on the Macintosh, the user assembly language program will run on the ECB. So, there has to be a way of reaching its internal registers, its memory etc. These are all ECB-related events. That is why the monitor program was implemented. The whole code is written assembly language. When the ECB is powered up, or it is Reset, the monitor program initializes the system and waits for a Macintosh command. During initialization phase, EPROM contents are copied to RAM, stack pointers are initialized, and program execution is switched to RAM.

After the initialization phase, the monitor simply loops endlessly, awaiting a command from the Macintosh. To provide an efficient way of receiving commands and processing them, there is a certain protocol established between the ECB and the Macintosh (detailed information about this protocol is given in Appendix B). According to this protocol, each command has a distinct one-byte-long code. Upon receiving this code, its corresponding command (such as memory write/display, etc.) is executed. If the command is a **Go** command, program execution continues at a user-provided program counter value. Following the execution of the user command, the monitor

continues to loop, waiting for the next command.

Among the functions of the monitor routine are providing communication with the Macintosh and supporting an Abort option. These two functions are briefly pointed out in the following two subsections. Furthermore, a detailed information about monitor program is given in Appendix C.

1. Serial Communication in Software

This debugger, as it was mentioned before, consists of two separate programs running on two different systems, namely the Macintosh and the ECB. This requires communication between the two. The protocol provided is typical of serial communication (at a Baud rate of 9600). As far as the ECB is concerned, this could be done in hardware, with the utilization of commercially available integrated circuits. The other possible choice was to do this in software (further information can be obtained from Ref. 4). The software approach was selected, slightly simplifying the hardware.

On the Macintosh side, a modem port is used for serial communication with the ECB. The modem port is initialized to a Baud rate of 9600 by download.c. In order to send or receive bytes, already available system calls are utilized. On the ECB side, two routines are written to provide serial communication with the Macintosh. RUART and SUART are the routines which provide communication outside of the ECB. RUART receives the incoming bytes via the RS232 input. SUART, on the other hand, transmits bytes via the RS232 interface. Both routines work at a Baud rate of 9600. This Baud rate is established by a clock frequency of 16 MHz. When the frequency is halved, for instance, so is the Baud rate.

When transmitting data at a Baud rate of 9600, the bits are 104.7 microseconds apart. Starting with the clock cycles needed to execute some of the instructions, an estimate of what instructions to use, how many times to loop in order to establish enough delay for a Baud rate of 9600, is made. Then, using the estimated values, an

approximate delay was obtained. Later on, by trial and error, enough delay is provided for a Baud rate of 9600.

In order to receive the incoming bytes, the RS232 input has to be sampled once every 104.7 microseconds. In this way, every single bit can be sensed. Briefly, the reception of a byte is done as follows: After detecting the start bit, eight consecutive bits are received. The detection of two stop bits follows this. The eight bits constitute the byte to be received. In case the stop bit is not detected, a frame error occurs. This reveals that an error is made during data transmission.

Sending of a byte on the other hand, starts with the transmission of the start bit. The transmission of eight bits follows this. It is provided that the duration of each bit is 104.7 microseconds. Finally the stop bits are sent. This action concludes the transmission of a single byte.

A detailed information about the communication routines is given in Appendix D.

2. Implementation of Abort Option

Another thing to be noted here is the Abort option. An Abort occurs when the abort button on the ECB is pressed. The Abort button is pressed in order to recover from an undesired situation. This undesired situation can be an endless loop, for example.

Pressing the Reset button also provides a recovery, but in this case all the register contents are lost, whereas pressing the Abort button causes a special Abort handler routine to execute which uploads all the current register contents to the Macintosh. As a result, the user can see all the register contents when he presses the Abort button. More information about Abort can be obtained from Appendix E.

IV. VALIDATION OF THE DEBUGGER

As the modules which perform the functions of the debugger were being developed, each was tested for correctness. After providing serial communication between the Macintosh and the ECB, the **download**, **memory display**, and **memory write** functions were implemented and tested. These three functions were then used in developing other functions of the debugger on the ECB side. Until the completion of these functions, the HP1650A logic analyzer was the only tool. After implementing all debugger functions, the overall debugger was exhaustively tested. Five sets of test programs were written, where each set tested the following sections of code.

- Communication between the Macintosh and the ECB.
- Downloading a program from Macintosh to the ECB.
- Displaying and modifying the ECB's memory/registers.
- Debugging MC68020 microprocessor instructions.
- Debugging MC68881 coprocessor instructions.

The instructions in this programs were selected such that they could test almost every possibility of a bug in the system (e.g., loss of stack space, modifying the memory incorrectly, etc.)

In the following two sections of this chapter the functionality of this debugger will be demonstrated by showing the results of test programs in response to the execution of the debugger commands. Three different test programs are debugged under various levels of control.

A. DEBUGGING MC68020 INSTRUCTIONS

In this section, a test program is demonstrated which contains various MC68020 instructions. Various levels of control (such as tracing, setting a breakpoint, etc.) were used while running the test program. In addition to testing these capabilities, some other debugger functions, such as download, memory write, and memory display are tested. For testing purposes, test program #1 was written in the file test.c. This is shown in Figure 1.

```
        MOVE.L    #0,D0; [D0] <- 0
        MOVE.L    #1,D1; [D1] <- 1
        MOVE.L    #2,D2; [D2] <- 2
        MOVE.L    #0,A0; [A0] <- 0
        MOVE.L    #1,A1; [A1] <- 1
        MOVE.L    #2,A2; [A2] <- 2
        MOVE.L    #3,D3; [D3] <- 3
        MOVE.L    #4,D4; [D4] <- 4
        MOVE.L    #5,D5; [D5] <- 5
L1:     SUB.L     #1,D5; [D5] <- [[D5]-1]
        BNE       L1     ;
        MOVEM.L   D0-D2,-(SP);
        MOVEM.L   (SP)+,D0-D2;
        ADD.L     #2,D2; [D2] <- [[D2]+2]
        TRAP      #15
```

Figure 1 Test program #1

This program is downloaded to the ECB, starting at the address $1000 (Dollar sign means hexadecimal). This is done by selecting the **Download** function in the main menu. The execution of the **memory display** command (displaying the memory

locations $1000 through $1032) shows the memory contents just before downloading the test program. In order to be able to execute the **memory display** command, the user needs to select the **Memory Display Menu** (see Figure 2). In Figure 2 the addresses ***From*** and ***To*** determines the portion of memory which is going to be displayed.

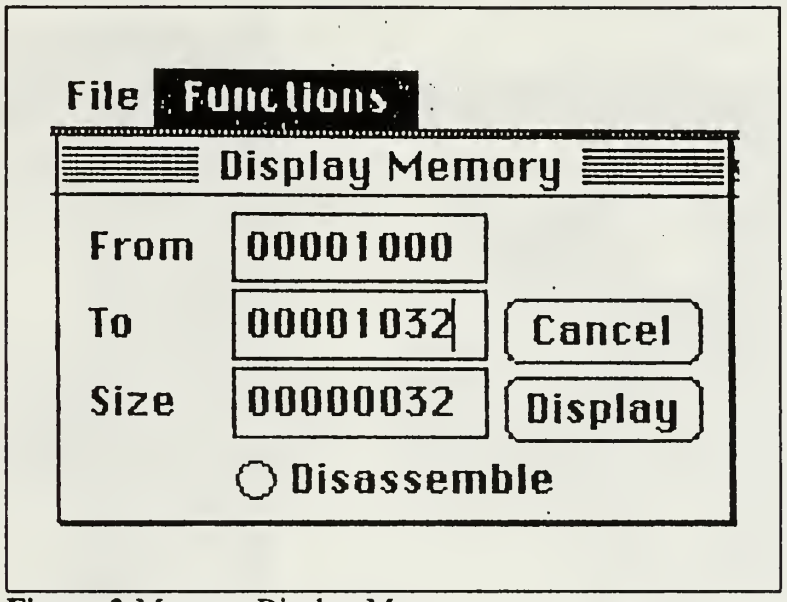


Figure 2 Memory Display Menu

As a result of the execution of the **memory display** command the following was obtained.

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00001000 FF FF FF EF FF BF FE EF FF FF FF FB FF EF EF FB  ................
00001010 FF FF FF BF FF FB FF FE FF EF FF EB FE FF FF FF  ................
00001020 FF FF EF BF FF FF FF EF FF FF FF FF FF FF FF FE  ................
00001030 FF FF BF xx xx xx xx xx xx xx xx xx xx xx xx xx  ...
```

The execution of the **download** command modifies the above memory contents. The new memory contents are the user program. This can be seen by executing the **memory display** command.

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00001000 70 00 72 01 74 02 20 7C 00 00 00 00 22 7C 00 00  p.r.t. |....'|..
00001010 00 01 24 7C 00 00 00 02 76 03 78 04 7A 05 53 85  ..$|....v.x.z.S.
00001020 66 00 FF FC 48 E7 E0 00 4C DF 00 07 54 82 4E 4F  f...H...L...T.NO
00001030 FF FF BF xx xx xx xx xx xx xx xx xx xx xx xx xx  ...
```

In order to see that the test program was correctly loaded the, **memory display** command was executed with the **disassemble** option. The result is shown in Figure 3.

```
00001000   7000                MOVEQ.L  #0,D0
00001002   7201                MOVEQ.L  #1,D1
00001004   7402                MOVEQ.L  #2,D2
00001006   207C00000000        MOVE.L   #0,A0
0000100C   227C00000001        MOVE.L   #1,A1
00001012   247C00000002        MOVE.L   #2,A2
00001018   7603                MOVEQ.L  #3,D3
0000101A   7804                MOVEQ.L  #4,D4
0000101C   7A05                MOVEQ.L  #5,D5
0000101E   5385                SUBQ.L   #1,D5
00001020   6600FFFC            BNE.L    $00101E
00001024   48E7E000            MOVEM.L  D0-D2,-(A7)
0000102B   4CDF0007            MOVEM.L  (A7)+,D0-D2
0000102C   5482                ADDQ.L   #2,D2
0000102E   4E4F                TRAP     #15
00001030   FFFF                WORD     $FFFF
00001032   BFBF                WORD     $BFBF
```

Figure 3 Disassembled test program #1

Then various levels of control were used during the execution of the test program. The first three commands in the test program were executed with the selection of the **Trace All** option in **Go menu.** (Go menu is shown in Figure 4.) With the **Trace All** option, the program execution returns to the debugger after the execution of every instruction. The debugger displays the result of each executed instruction.

Setting the program counter value to $1000, selecting the **Trace All** option, and then clicking go causes the first instruction of the test program #1 to be executed. The

result is shown below.

```
PC=00001002
SR=8004     USP=0001F800 SSP=0001FC00 ISP=0001FFFC
D0=00000000  D1=00000000  D2=00000000  D3=00000000
D4=00000000  D5=00000000  D6=00000000  D7=00000000
A0=00000000  A1=00000000  A2=00000000  A3=00000000
A4=00000000  A5=00000000  A6=00000000  A7=0001F800
00001002    7201                       MOVEQ.L  #1,D1
```

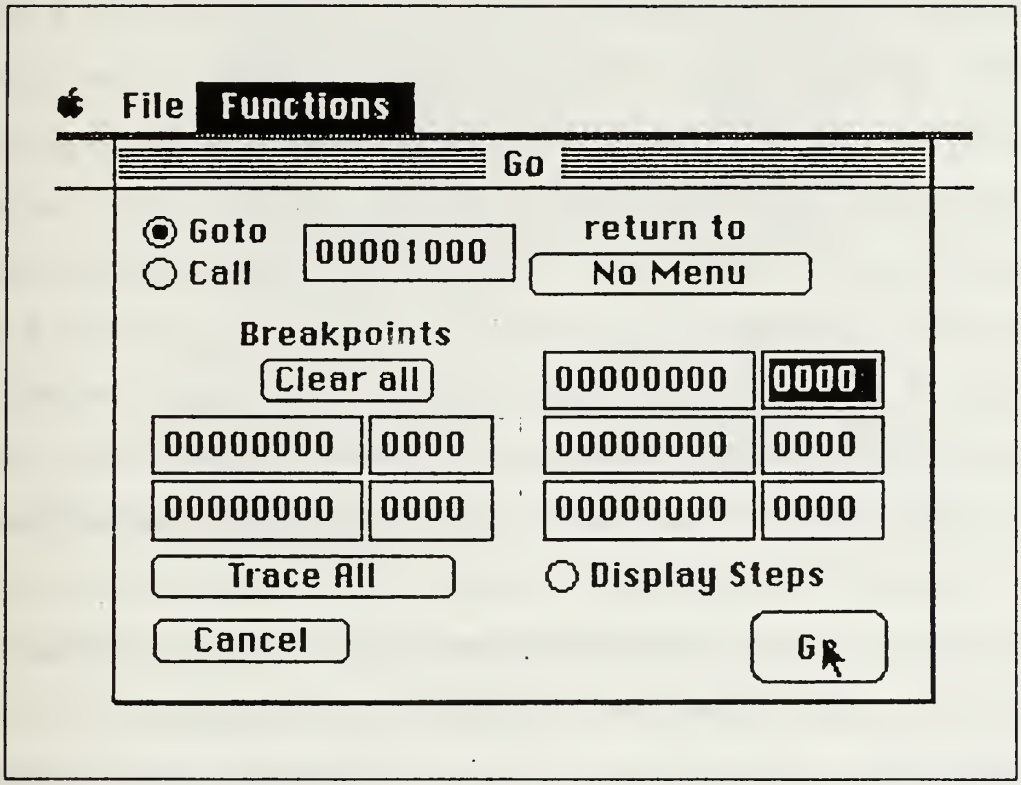


Figure 4 Go menu

Clicking go once more gives the following:

```
PC=00001004
SR=8000     USP=0001F800 SSP=0001FC00 ISP=0001FFFC
D0=00000000  D1=00000001  D2=00000000  D3=00000000
D4=00000000  D5=00000000  D6=00000000  D7=00000000
A0=00000000  A1=00000000  A2=00000000  A3=00000000
A4=00000000  A5=00000000  A6=00000000  A7=0001F800
00001004    7402                       MOVEQ.L  #2,D2
```

And finally, clicking go once again, one more instruction is executed.

```
PC=00001006
SR=8000      USP=0001F800 SSP=0001FC00 ISP=0001FFFC
D0=00000000  D1=00000001  D2=00000002  D3=00000000
D4=00000000  D5=00000000  D6=00000000  D7=00000000
A0=00000000  A1=00000000  A2=00000000  A3=00000000
A4=00000000  A5=00000000  A6=00000000  A7=0001F800
00001006     207C00000000              MOVE.L   #0,A0
```

Before changing the level of control, looking at the outcomes of the previous three steps, it is seen that three instructions were executed correctly. That is, as a result of the MOVE instructions, the new contents of the data registers D0, D1, and D2 are zero, one, and two respectively. The results of the instructions could also be seen by selecting the registers menu (in the main menu). The registers menu can also be displayed without going through the main menu. The format of the information which is displayed after the execution of the user program depends on the selection of the ***return to*** option in Go menu. The user has three choices. When return to is selected as Go menu, following the execution of user program, Go menu is displayed again. When return to is selected as No menu, no menu is displayed on the screen, instead register contents are displayed. (This is the format used in the previous three steps.) And as the third choice, return to can be selected as Registers menu. In this case, following the execution of user program, Registers menu is displayed on the screen. In order to see this, the last trace step is repeated with return to selected as ***Registers menu***. The result is shown in Figure 5.

In order to see the effect of the **Trace Branch** option, consider the following. With the **Trace Branch**, program execution returns to the debugger when a branch (either the unconditional branch BRA or one of the conditional branches, such as BEQ, BNE, etc.) is taken. This means that the user will be able the see the results when

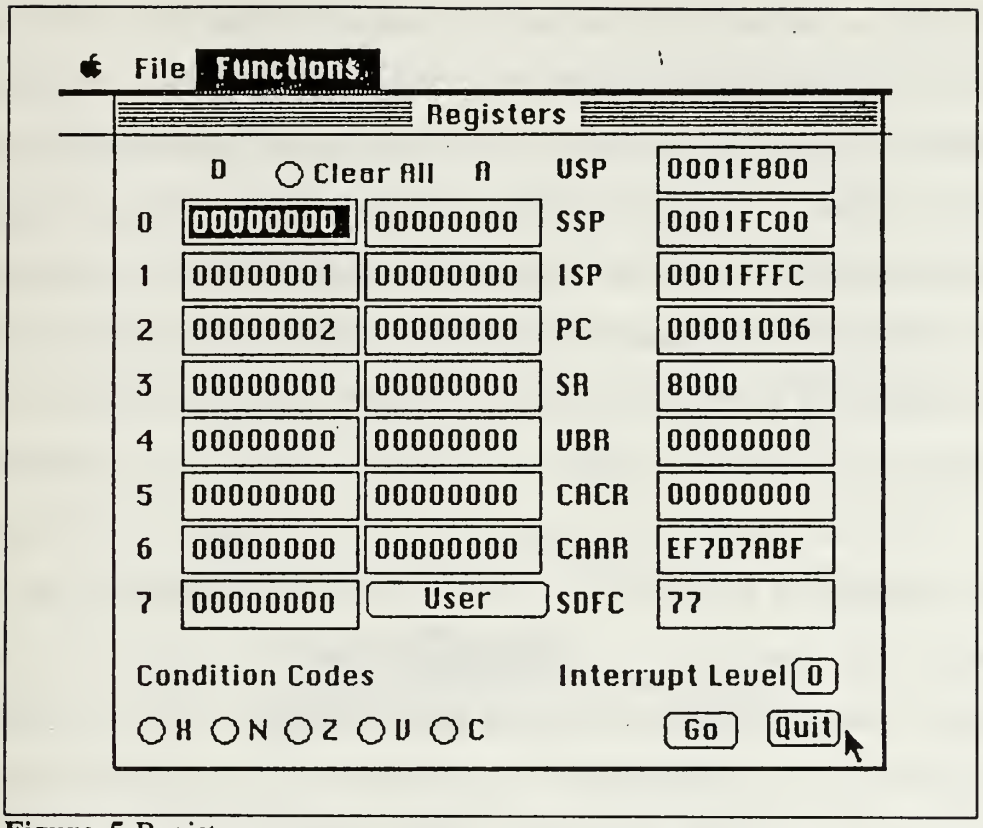


Figure 5 Registers menu

there is a change on the flow of the program. In the test program #1 of Figure 1, there is a branch instruction BNE which executes five times. The expected result after one execution is that, when the registers are displayed on the screen, the program counter content is $101E, and data register D5 contains four (since it is loaded with five and is decremented by one before the branch).

```
PC=0000101E
SR=6000      USP=0001F800 SSP=0001FC00 ISP=0001FFFC
D0=00000000  D1=00000001  D2=00000002  D3=00000003
D4=00000004  D5=00000004  D6=00000000  D7=00000000
A0=00000000  A1=00000001  A2=00000002  A3=00000000
A4=00000000  A5=00000000  A6=00000000  A7=0001F800
0000101E     5385                      SUBQ.L   #1,D5
```

The previous output was exactly the same as expected. Now the use and effect of a **breakpoint** is illustrated. In test program #1 of Figure 1 there are two instructions which perform a push onto the stack and a pop from the stack. After executing these instructions, the content of the stack pointer should remain unchanged. Before executing, the trace level was set to **No Trace** and a **breakpoint** was set to the address $1028. There are three stack pointers in the MC68020. They are: User Stack Pointer (USP), Supervisor Stack Pointer (SSP), and the Interrupt Stack Pointer (ISP). In Go menu the default stack pointer is the USP (the active stack pointer can be changed to another one by the user). So, the stack operations in the test program #1 will be in the User Stack. The instruction MOVEM.L D0-D2,-(SP) will push the registers D0, D1, D2 onto the stack. At the breakpoint the displayed USP content is expected to be 12 less than its original value (as a result of the pushes onto the stack). And also the program counter should point to the instruction at the breakpoint address. The following output was obtained after this step. By examining the register contents, it was verified that the result is correct.

```
PC=00001028
SR=0004      USP=0001F7F4 SSP=0001FC00 ISP=0001FFFC
D0=00000000  D1=00000001  D2=00000002  D3=00000003
D4=00000004  D5=00000000  D6=00000000  D7=00000000
A0=00000000  A1=00000001  A2=00000002  A3=00000000
A4=00000000  A5=00000000  A6=00000000  A7=0001F7F4
00001028     4CDF0007                  MOVEM.L  (A7)+,D0-D2
```

At this point, the use of the **memory display** command (of memory locations $1F7E0 through $1F7FF) shows the new stack contents.

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0001F7E0 FF FF FF FF FF FF FF FF FF FF F7 DF FF FF FF D7  ................
0001F7F0 FF FF DF FF 00 00 00 00 00 00 00 01 00 00 00 02  ................
```

The underlined part of the previous output shows that the contents of data registers D0, D1, D2 are pushed on to the stack. The longword (four bytes) at the address $1F7F4 contains the content of D0, the next longword (at the address $1F7F8) contains the content of D1, and the longword at the address $1F7FC contains the content of D2. This current stack content can be changed with the use of the **memory write** command. In order to be able to execute the **memory write** command, the user needs to select the **Memory Write Menu** in the main menu. This menu is shown in Figure 6.

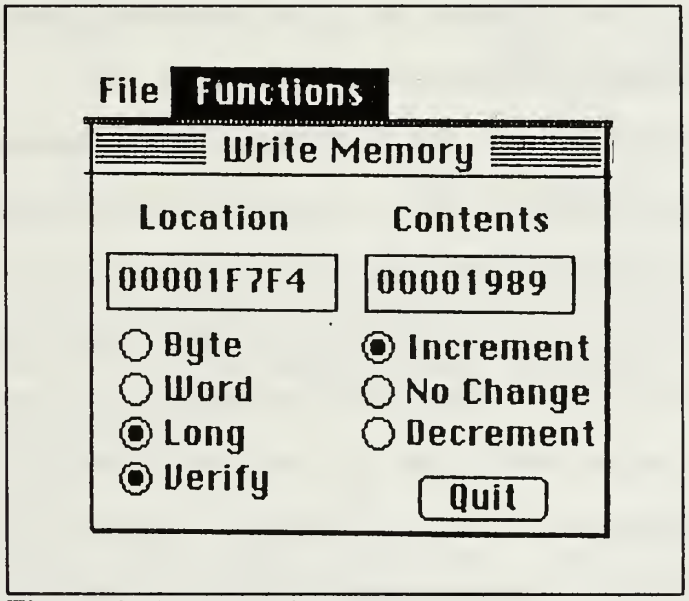


Figure 6 Memory write menu

By executing the **memory write** command (writing $00001989 to the address $1F7F4), and, following that, executing a **memory display** command, the stack contents become:

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0001F7E0 FF FF FF FF FF FF FF FF FF FF F7 DF FF FF FF D7  ................
0001F7F0 FF FF DF FF 00 00 19 89 00 00 00 01 00 00 00 02  ................
```

This last execution shows the effect of the **memory write** command. Up to this point the test program has been executed either by **tracing** or by setting a **breakpoint**. If there are no trace or breakpoints in the program, execution returns to the debugger when the final instruction (Trap #15) is encountered. In order to test this, breakpoints were removed and **No Trace** option was selected before clicking **go**.

```
PC=0000102E
SR=0000      USP=0001F800 SSP=0001FC00 ISP=0001FFFC
D0=00000000  D1=00000001  D2=00000004  D3=00000003
D4=00000004  D5=00000000  D6=00000000  D7=00000000
A0=00000000  A1=00000001  A2=00000002  A3=00000000
A4=00000000  A5=00000000  A6=00000000  A7=0001F800
0000102E     4E4F                      TRAP     #15
```

As it is seen in the last output, the program execution is indeed returned to the debugger when the **Trap #15** instruction was encountered.

The following program (test program #2) is exactly the same as the test program which has already been described with the exception of **Trap #15** instruction which is now replaced by an **RTS** instruction. As far as the execution of the user program is concerned, there are two modes in **Go menu**. They are: **Goto** and **Call**. Test program #1 was executed with the mode **Goto** (this is the default mode) selected in **Go menu**. **Call** option is provided in order to test the subroutines. When Call is selected as the operating mode, after the execution of the subroutine the program counter points to the beginning address of the subroutine just called (for more details see Appendix F). In order to illustrate the use of mode **Call**, test program #2 was written in the file test.c, the debugger was run and the program was downloaded to the ECB. Execution of a **memory display** command (displaying the memory locations $1000 through $1032) with the **disassemble** option displays test program #2 and verified the correctness of downloading. This is shown in Figure 7.

```
00001000    7000                    MOVEQ.L  #0,D0
00001002    7201                    MOVEQ.L  #1,D1
00001004    7402                    MOVEQ.L  #2,D2
00001006    207C00000000            MOVE.L   #0,A0
0000100C    227C00000001            MOVE.L   #1,A1
00001012    247C00000002            MOVE.L   #2,A2
00001018    7603                    MOVEQ.L  #3,D3
0000101A    7804                    MOVEQ.L  #4,D4
0000101C    7A05                    MOVEQ.L  #5,D5
0000101E    5385                    SUBQ.L   #1,D5
00001020    6600FFFC                BNE.L    $00101E
00001024    48E7E000                MOVEM.L  D0-D2,-(A7)
00001028    4CDF0007                MOVEM.L  (A7)+,D0-D2
0000102C    5482                    ADDQ.L   #2,D2
0000102E    4E75                    RTS
00001030    FFFF                    WORD     $FFFF
00001032    FFBF                    WORD     $FFBF
```

Figure 7 Disassembled test program #2

Before clicking go, the program counter value is set to $1000. The output after the execution is shown below. The program counter still points to $1000 after the execution of the test subroutine.

```
PC=00001000
SR=0000      USP=0001F800 SSP=0001FC00 ISP=0001FFFC
D0=00000000  D1=00000001  D2=00000004  D3=00000003
D4=00000004  D5=00000000  D6=00000000  D7=00000000
A0=00000000  A1=00000001  A2=00000002  A3=00000000
A4=00000000  A5=00000000  A6=00000000  A7=0001F800
00001000     7000                      MOVEQ.L  #0,D0
```

B. DEBUGGING MC68881 INSTRUCTIONS

In this section we consider the verification of coprocessor-related capabilities of the debugger. For this purpose, test program #3 was written. This test program contains

two coprocessor instructions and is shown in Figure 8. Prior to execution floating point register FP4 is assumed to contain a number X whose sine and cosine are to be computed.

```
DC.W        $F200; FSINCOSX.X FP4,FP5,FP6
DC.W        $12B6;
DC.W        $F23C; FMOVE.L #7,FP7
DC.W        $4380;
DC.W        $0000;
DC.W        $0007;
```

Figure 8 Test program #3

This program was written in test.c, the debugger was run, **coprocessor** option was selected in **go menu**, and test program was downloaded to the ECB. The result of executing the **memory display** command (disassembling the memory contents starting from $1000 and ending at $1014) is shown in Figure 9. Since the current disassembler is not able to disassemble coprocessor-related instructions, these unsupported instructions are displayed in their hexadecimal representation. In this test, 0.785375 was entered in the register FP4 as X (0.785375 radians corresponds to 45 degrees). Following this, the program counter value was set to $1000, and **go** was clicked. The expected result is the sine of 45 degrees (which is nearly 0.707) in floating point register FP5, the cosine of 45 degrees in FP6 (which is also 'nearly 0.707), and of course 0.785375 in FP4. As a result of the second instruction, floating point register FP7 was supposed to contain seven. The outcome of this test run is shown in Figure 9

where the floating point registers menu is displayed. As it is seen in Figure 9 the result is exactly the same as it was expected.

```
00001000    F200                    WORD    $F200
00001002    12B6                    WORD    $12B6
00001004    F23C                    WORD    $F23C
00001006    4380                    CHK.W    D0,D1
00001008    00000007                OR.B     #7,D0
0000100C    4E4F                    TRAP     #15
0000100E    F206                    WORD    $F206
00001010    4322                    WORD    $4322
00001012    4E4F                    TRAP     #15
00001014    00000002                OR.B     #2,D0
```

Figure 9 Disassembled test program #3

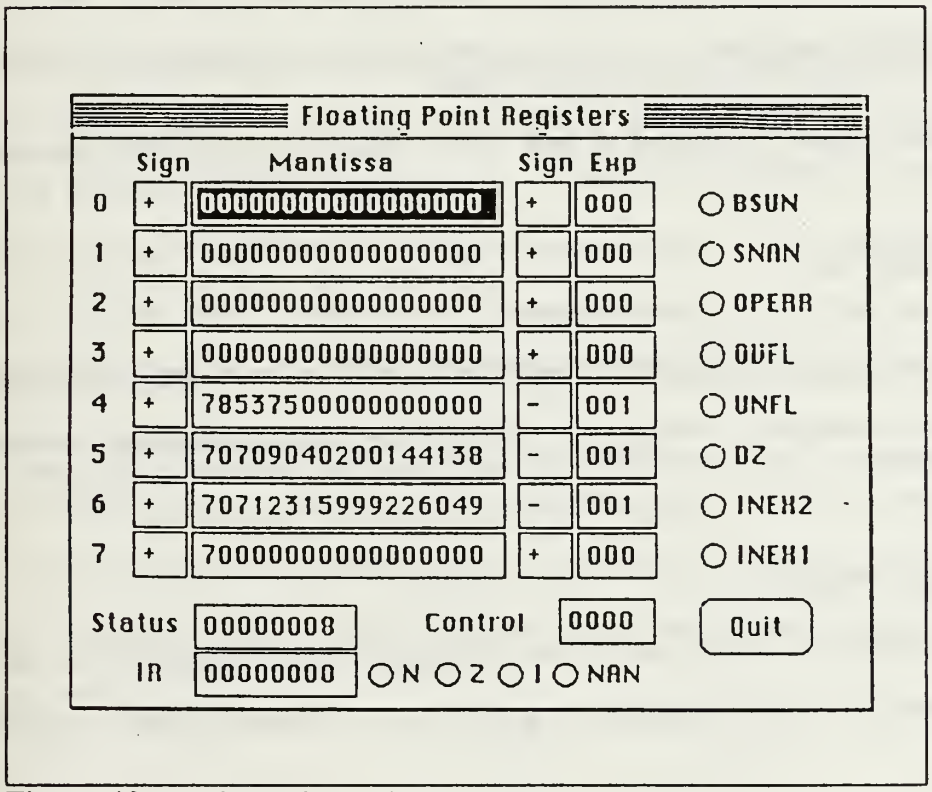


Figure 10 Floating point registers menu

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The debugger written under the topic of this thesis study can be considered as an up-to-date version of the Motorola's debugger, Tutor, which was in wide use for a long time. This debugger, together with the MC68020 based ECB, constitutes a very handy tool for students and for researchers. When compared to the Motorola's debugger, it has some advantages and some disadvantages.

The advantages are:

- It can support MC68020 state-of-the-art microprocessors rather than MC68000. It can handle Coprocessor instructions.
- The user does not have to memorize some debugger commands, using pull-down menus, it is easier to learn and easier to use.
- No dumb terminal is needed as part of the debugger. Instead the Macintosh is utilized as an intelligent front end.

The disadvantages are:

- Fewer debugger commands are supported compared to the Motorola's debugger.
- Since this debugger communicates with ECB via RS232 interface, which takes some amount of time, it is somewhat slower than the Motorola's debugger.

B. FUTURE WORK

As was mentioned before, part of the debugger resides in EPROM and runs on ECB. It is called the monitor program. In monitor, only a limited number of exceptions

could be supported due to limited amount of time for this thesis study. The exceptions which have associated exception handler routines are: Reset, Privilege Violation, Level 4 and Level 6 Autovectored Interrupts, Trace and Trap #15 (for more information about exceptions, see section 6 in Ref. 2). The other exception vector entries are loaded with the address of a short routine (STACKFRAME), which does nothing but arrange the stack. This prevents the loss of some stack space and system lock. As a future study, the corresponding exception handlers can be written for the yet unsupported exceptions.

By selecting **Disassemble** option, the desired memory locations can be disassembled and displayed on the screen. But the disassembly routine handles only MC68000 instructions. MC68020 instructions are not supported. They are displayed in their hexadecimal form. As a future work, with some additions to the disassembly routine, the disassembly of MC68020 instructions can be supported.

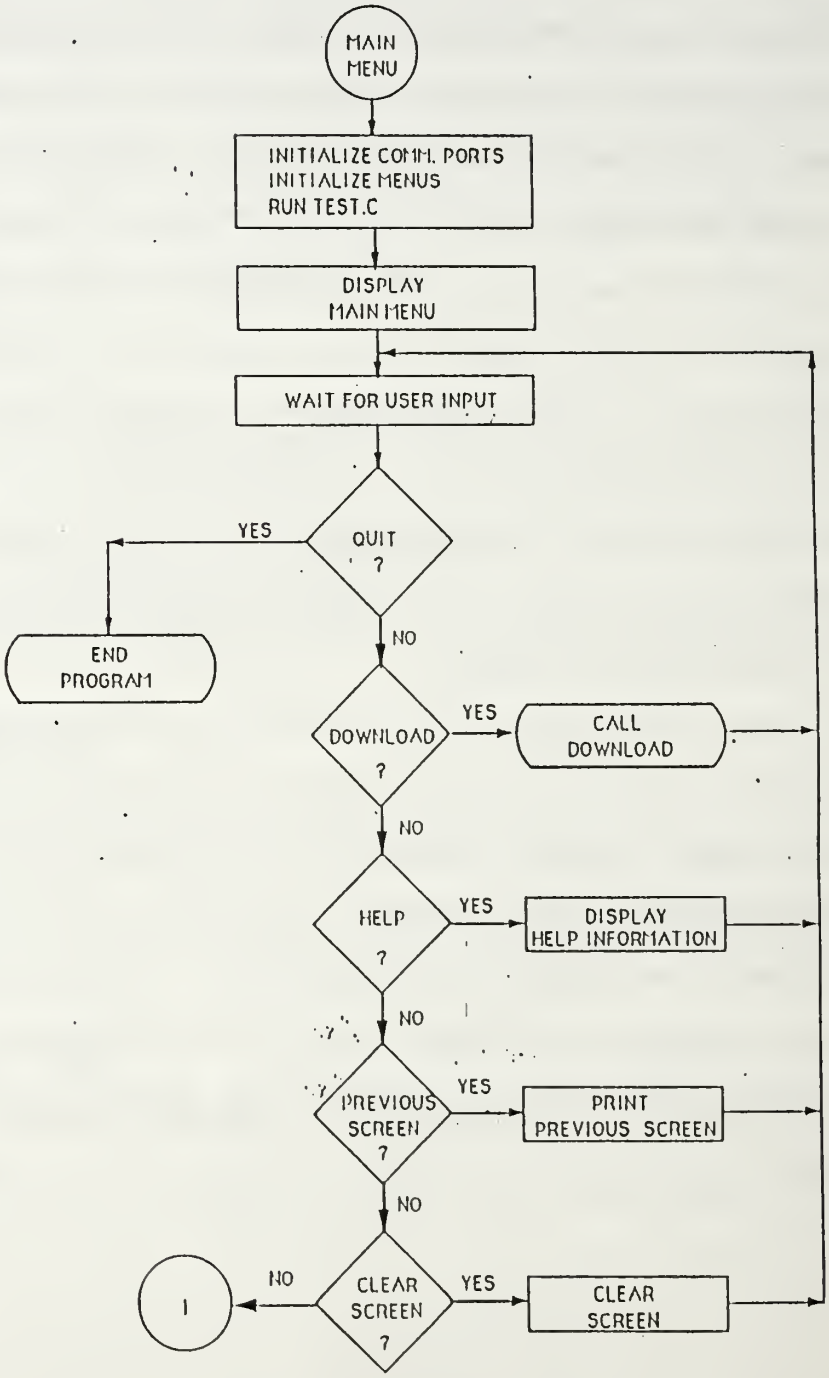
As another improvement to this debugger, some more debugger commands can be supported, which enable the user to **Fill a Block of Memory**, **Move a Block of Memory** or a **Search a Block of Memory**.

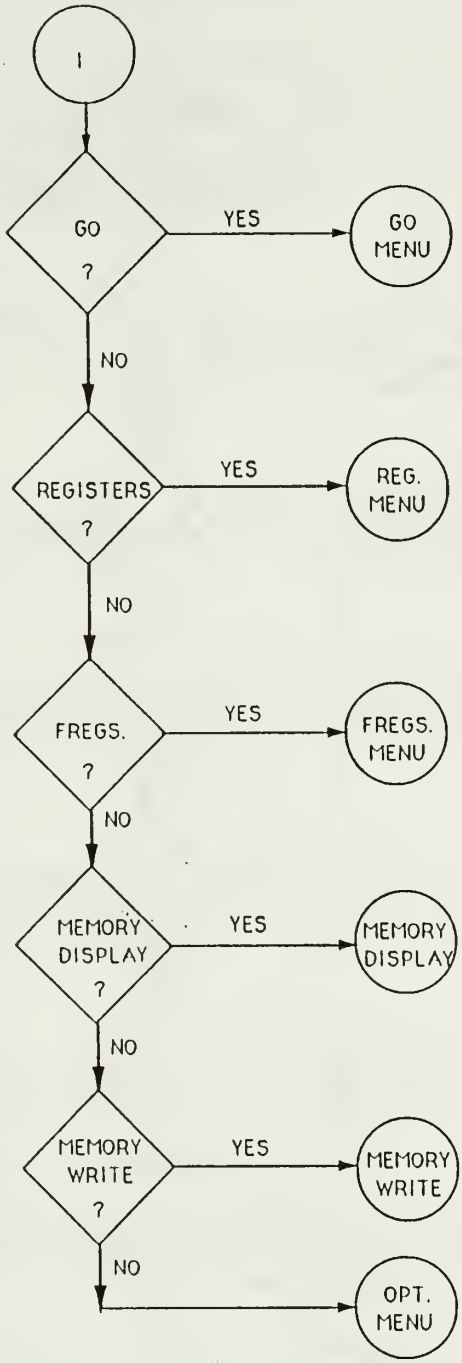
In the current version of this debugger, the program to be downloaded has to exist in a single file, test.c. It may be very beneficial if the user is given the option of downloading the program in any one of different files. This could not be done because current version of LightspeedC did not allow it.

This debugger has the capability of providing the user a hardcopy option. But it works only with Imagewriter serial printer. It will be very practical if a variety of Macintosh compatible printers can be included in a menu, where the user can select which one to use.

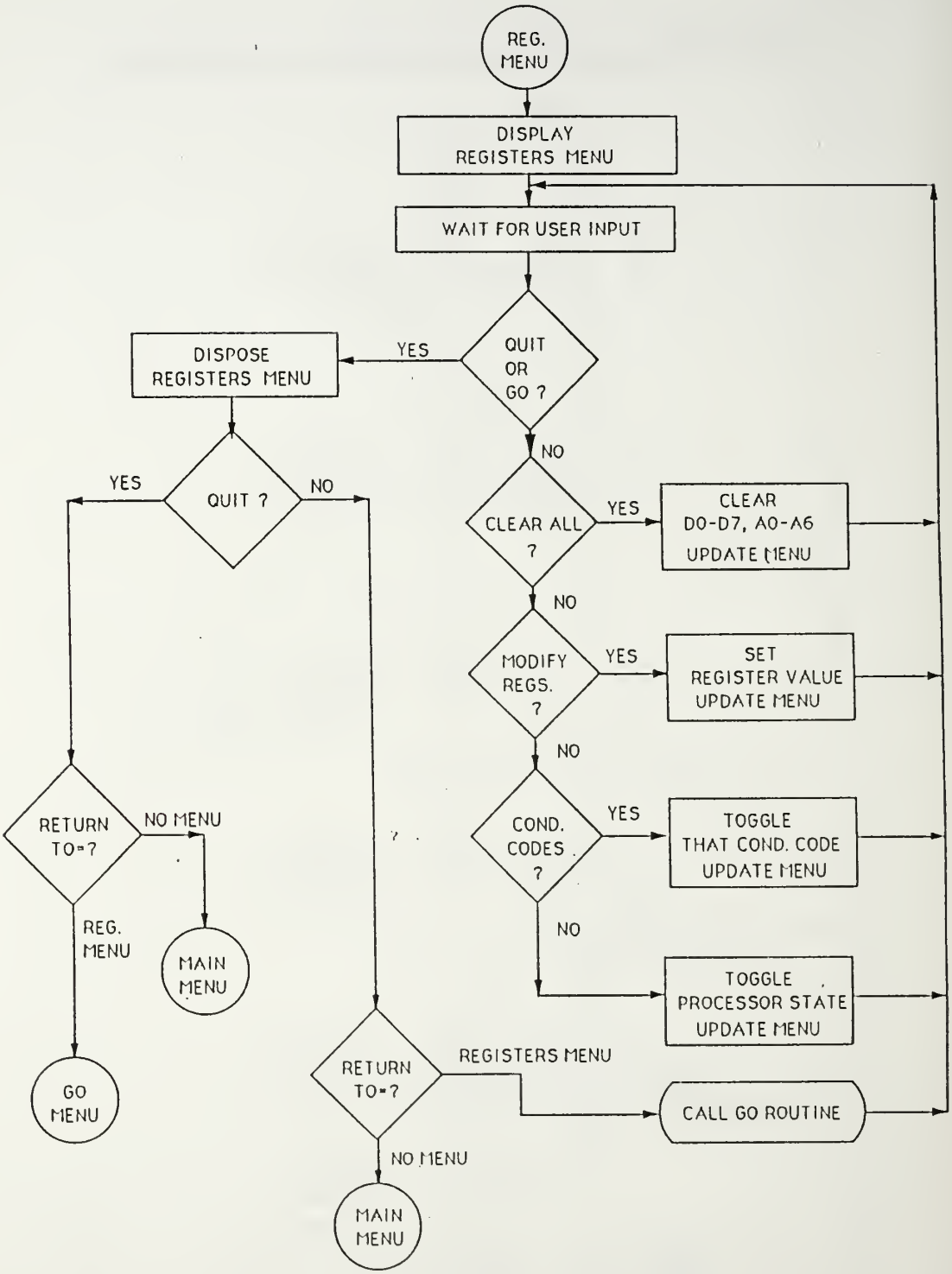
APPENDIX A: FLOWCHARTS FOR THE DEBUGGER

1 - FLOWCHART FOR MAIN MENU

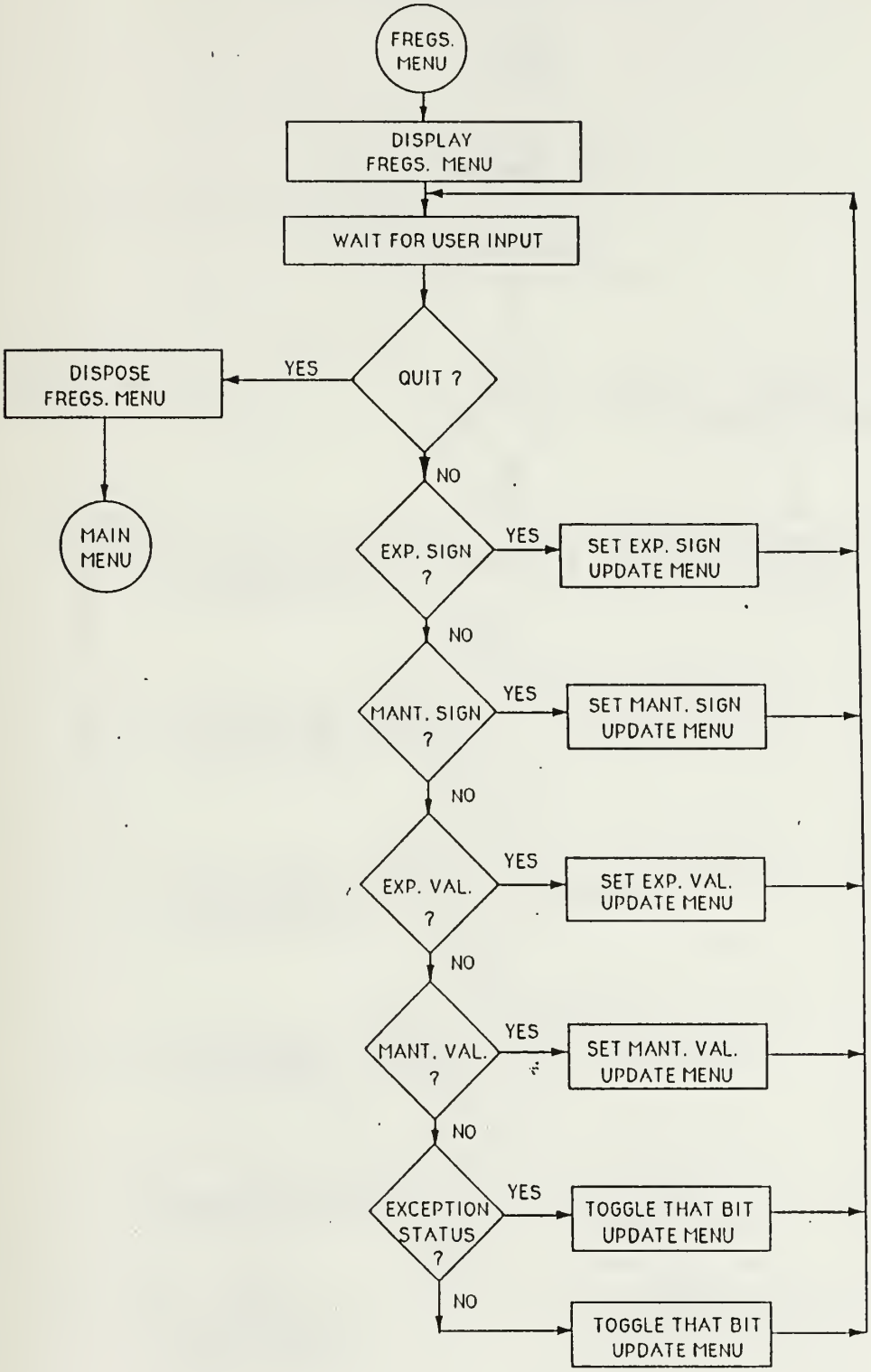



I
GO ?
YES
GO MENU
NO
REGISTERS ?
YES
REG. MENU
NO
FREGS. ?
YES
FREGS. MENU
NO
MEMORY DISPLAY ?
YES
MEMORY DISPLAY
NO
MEMORY WRITE ?
YES
MEMORY WRITE
NO
OPT. MENU

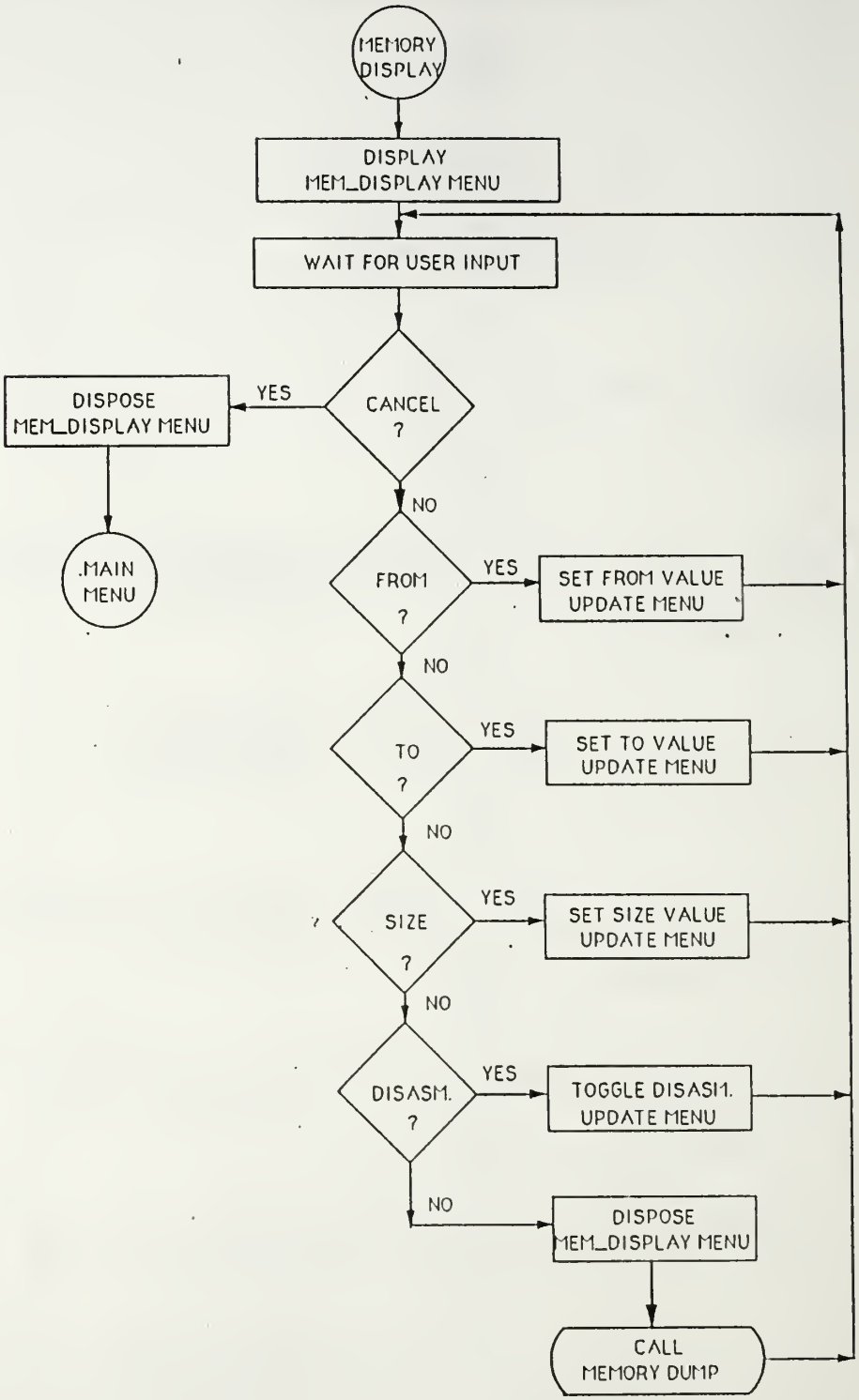
2- FLOWCHART FOR REGISTERS MENU



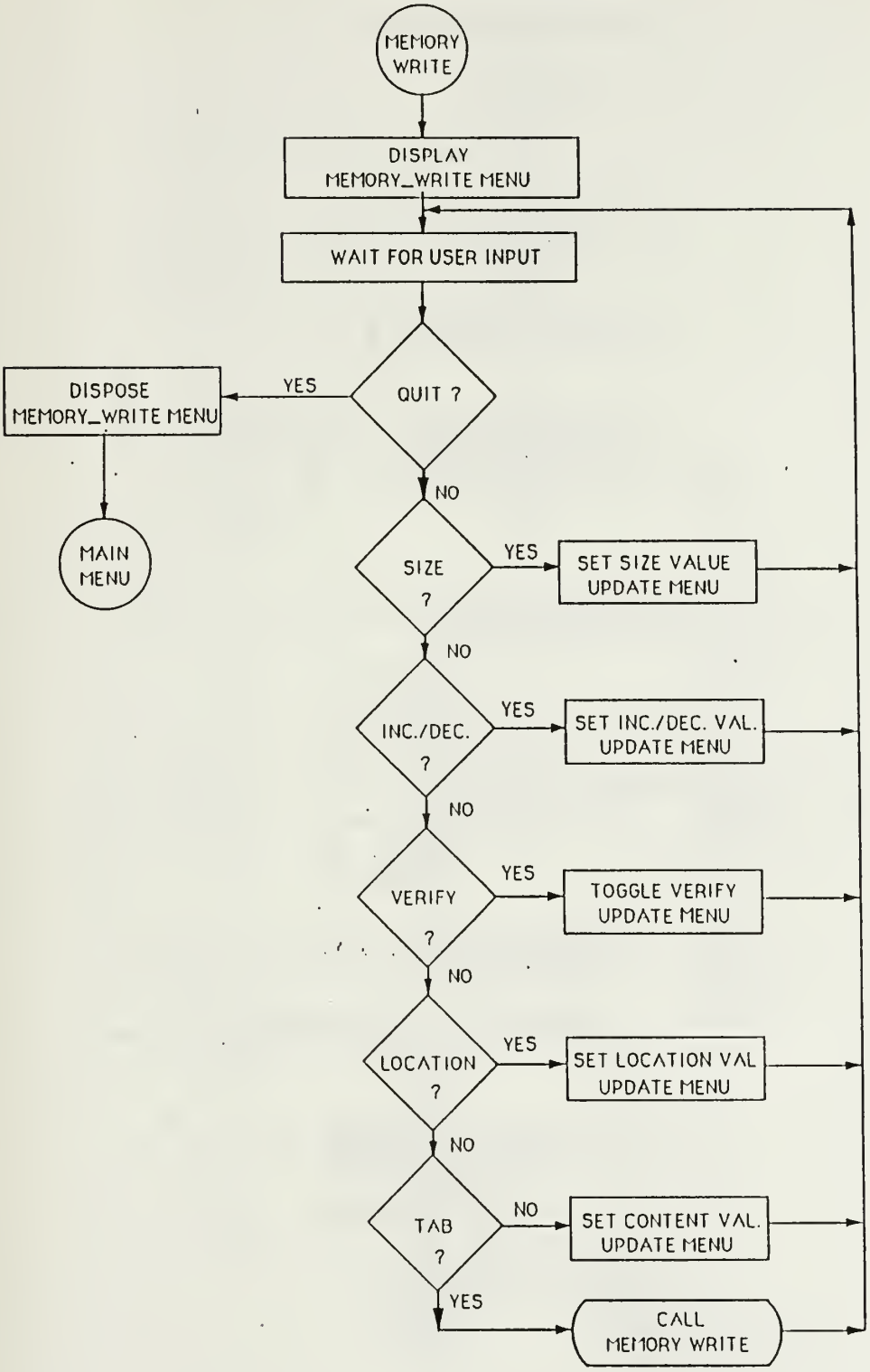
3- FLOWCHART FOR FLOATING POINT REGISTERS MENU



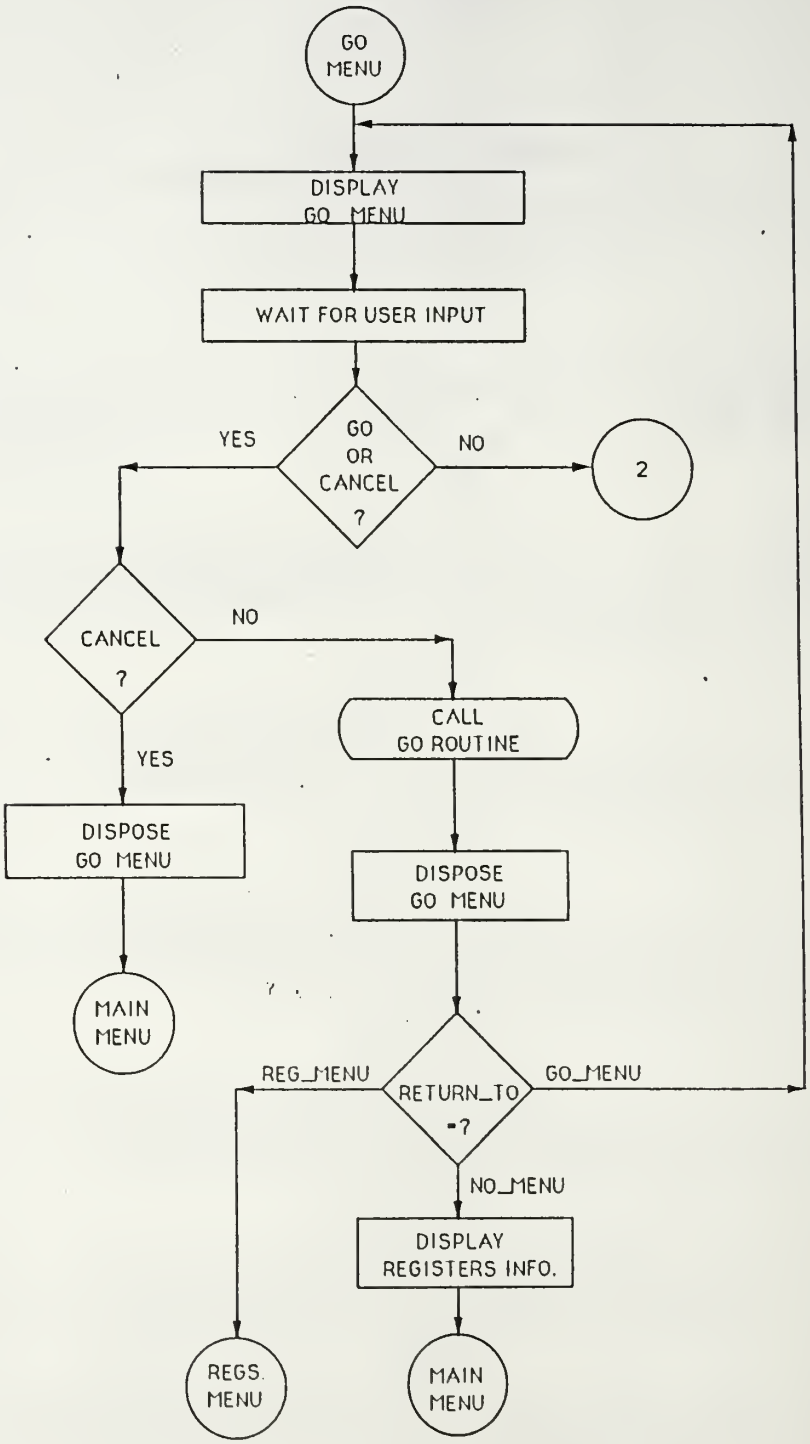
4- FLOWCHART FOR MEMORY DISPLAY MENU

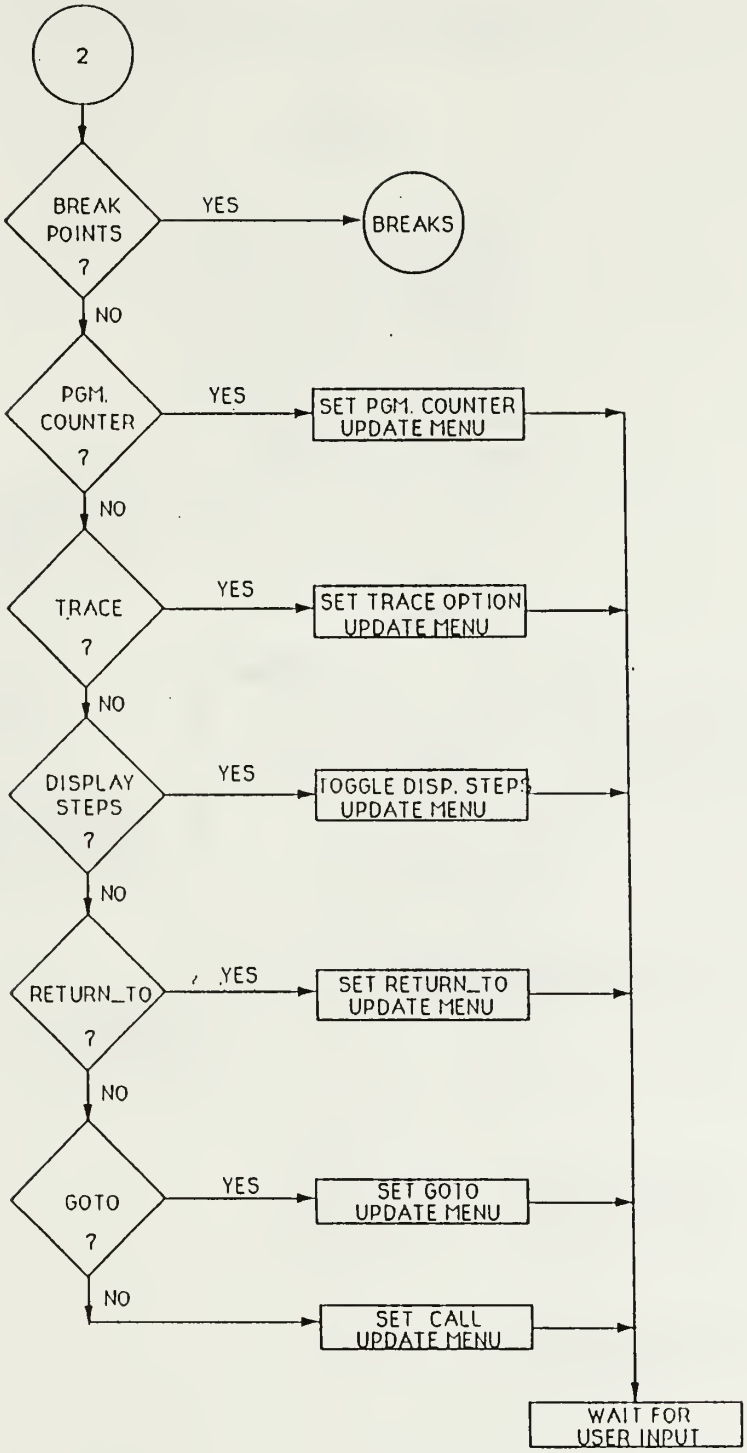


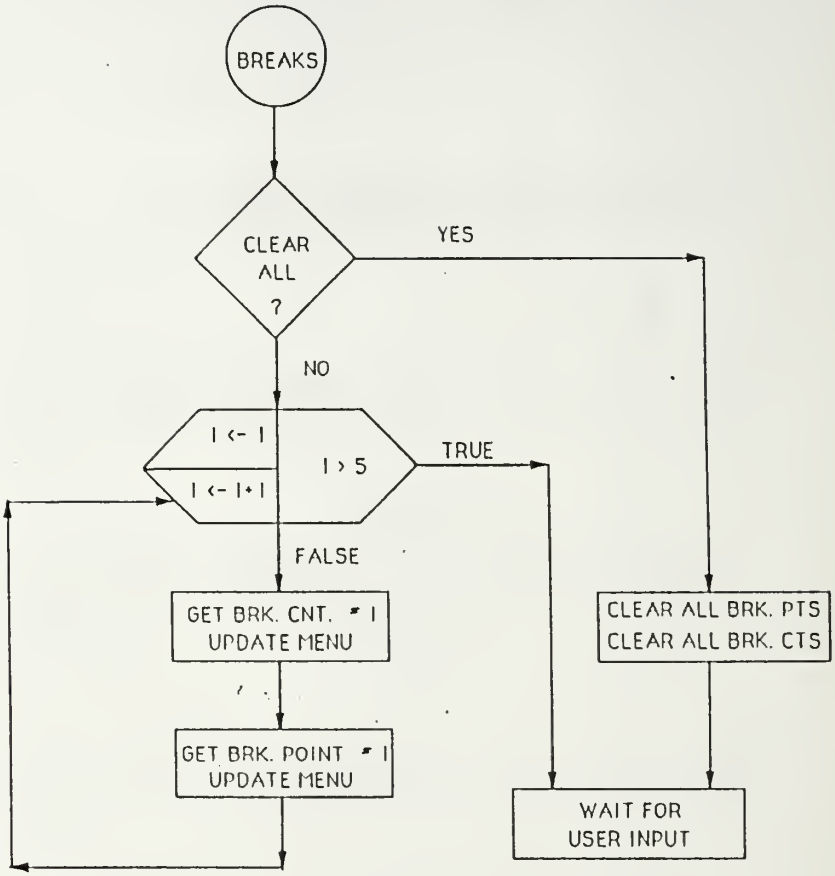
5- FLOWCHART FOR MEMORY WRITE MENU



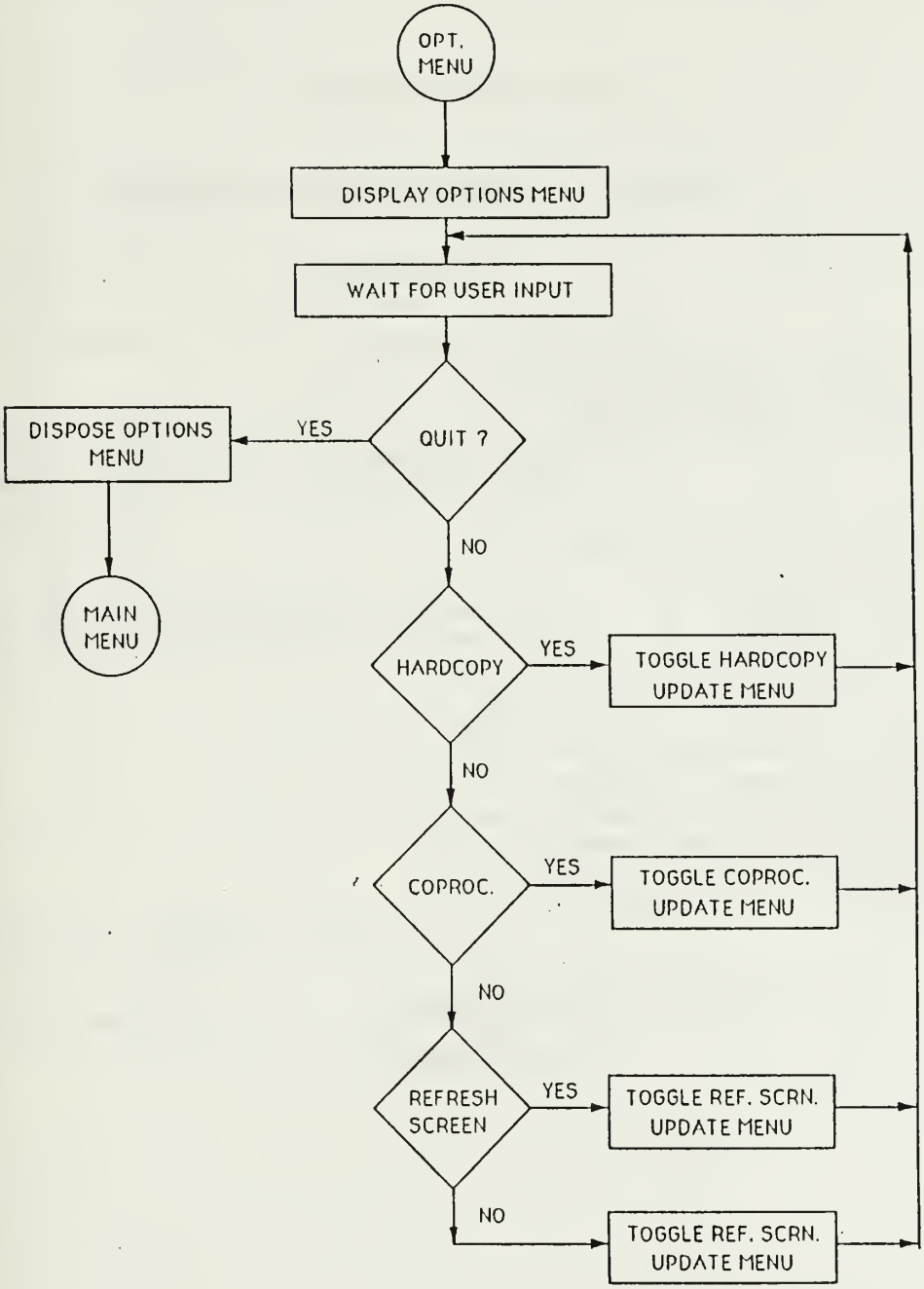
6- FLOWCHART FOR GO MENU



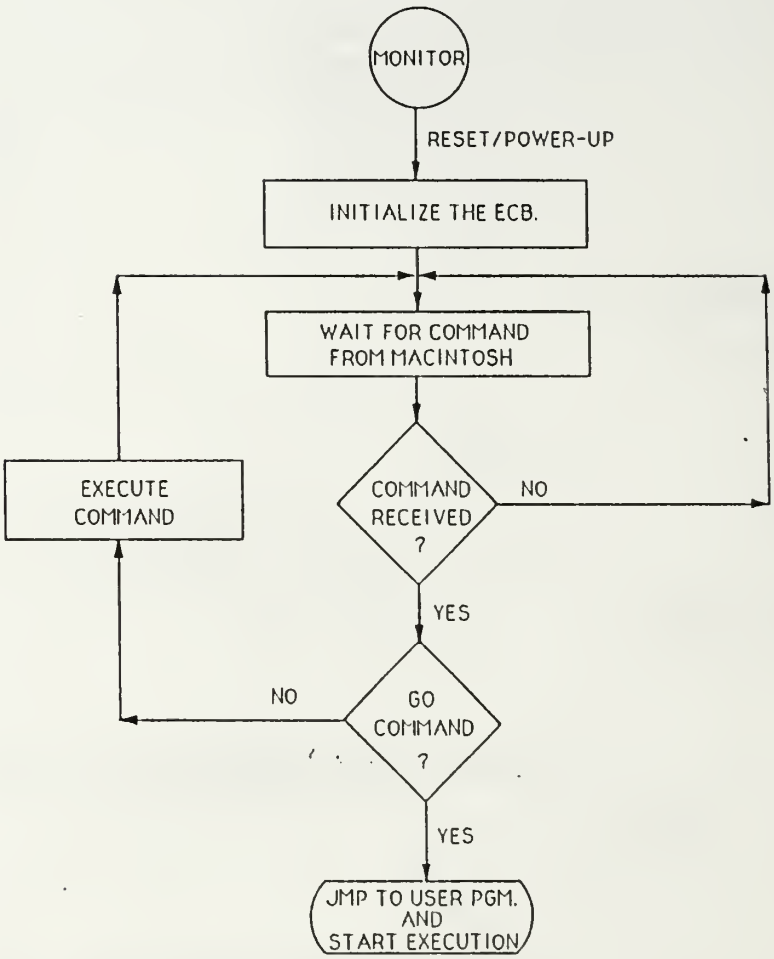

2
BREAK POINTS ?
YES
BREAKS
NO
PGM. COUNTER ?
YES
SET PGM. COUNTER UPDATE MENU
NO
TRACE ?
YES
SET TRACE OPTION UPDATE MENU
NO
DISPLAY STEPS ?
YES
TOGGLE DISP. STEPS UPDATE MENU
NO
RETURN_TO ?
YES
SET RETURN_TO UPDATE MENU
NO
GOTO ?
YES
SET GOTO UPDATE MENU
NO
SET CALL UPDATE MENU
WAIT FOR USER INPUT


BREAKS
CLEAR ALL ?
YES
NO
I <- 1
I <- I+1
I > 5
TRUE
FALSE
GET BRK. CNT. * I
UPDATE MENU
GET BRK. POINT * I
UPDATE MENU
CLEAR ALL BRK. PTS
CLEAR ALL BRK. CTS
WAIT FOR USER INPUT

7- FLOWCHART FOR OPTIONS MENU



8- FLOWCHART FOR MONITOR PROGRAM



APPENDIX B: MACINTOSH-ECB INTERFACE PROTOCOLS

EXECUTION OF MEMORY DISPLAY COMMAND

MACINTOSH	ECB
- Send memory_display_code.	
	1- Main receives memory_display_code, and switches program execution to MEMORY_DISPLAY Routine.
- Send the start_address (4 bytes) of the memory locations which are to be displayed.	
	2- Receive the start_address (4 bytes)
- Send byte_count (1 byte) which is the number_of_bytes to be displayed.	
	3- Receive byte_count (1 byte).
	4- Read from memory locations, starting from start_address, and send them to Macintosh, one-by-one. Meanwhile calculate the checksum byte. Checksum is calculated by EXORing the outgoing bytes.
- Receive memory contents (As many as byte_count bytes).	
- Receive checksum byte.	5- Send checksum byte.

EXECUTION OF MEMORY WRITE COMMAND

MACINTOSH	ECB
1- Send memory_modify_code.	1- Main receives memory_modify_code, and switches program execution to MEMORY_WRITE Routine.
2- Send the operand_size (width) byte (1 byte).	2- Receive the operand_size (1 byte).
3- Send the address of the memory location to be modified (4 bytes).	3- Receive the address of the memory location to be modified (4 bytes).
4- Send new memory contents (1, 2 or 4 bytes, depending on the width).	4- Receive new memory contents and write them into the memory locatio starting from memory_modify start address.
	5- [If "verify" option is selected] Read from memory locations and sen them to the Macintosh. (As many as byte_count bytes).
5- [If "verify" option is selected] Receive new memory contents. (As many as byte_count bytes).	

EXECUTION OF DOWNLOAD COMMAND

MACINTOSH	ECB
1- Send download_code.	
	1- Main receives download_code, and switches program execution to DOWNLOAD Routine.
2- Send the download_address. User program will be loaded starting from this address. (4 bytes).	
	2- Receive the download_address. (4 bytes).
3- Send the number_of_bytes to be downloaded. (2 bytes).	
	3- Receive the number_of_bytes. (2 bytes).
4- Send all the bytes which constitute the user program. Meanwhile calculate the checksum. (As many as number_of_bytes bytes will be sent).	
	4- Receive user program, byte by byte. Meanwhile calculate the checksum. (As many as number_of_bytes bytes will be received).
5- Send the checksum byte. (1 byte).	
	5- Receive the checksum byte. (The one sent by the Macintosh). (1 byte).
	6- Upload MC68020 register contents to Macintosh. (96 bytes). [If "Coprocessor" is selected] Upload MC68881 register contents to Macintosh. (108 bytes).
6- Receive MC68020 register contents. (96 bytes). [If "Coprocessor" is selected] Receive MC68881 register contents. (108 bytes).	

EXECUTION OF GO COMMAND

MACINTOSH	ECB
1- Send go_code.	
	1- Main receives go_code, and switches program execution to GO Routine.
2- Send Display_Steps (1 byte).	
	2- Receive Display_Steps (1 byte).
3- Send all the Break_Points, starting from Break_Point #0. (Four bytes per Break_Point, in total 20 bytes).	
	3- Receive Break_Point addresses. Four bytes per Break_Point. (20 bytes).
4- Send all the Break_Counts, starting from Break_Count #0. (Two bytes per Break_Count, in total 10 bytes).	
	4- Receive Break_Counts. (Two bytes per Break_Count, in total 10 bytes).
5- Send MC68020 register contents. D0-D7, A0-A6, and Control registers. (96 bytes).	
	5- Receive MC68020 register contents. D0-D7, A0-A6, and Control registers. (96 bytes).
6- Send checksum byte (1 byte).	
	6- Receive checksum byte (1 byte).
7- [If "Coprocessor" is selected] Send Coprocessor register contents.	
	7- [If "Coprocessor" is selected] Receive Coprocessor register contents. (108 bytes).
	8- Start the execution of user program. [When user program execution stops due to a "Trace", "Breakpoint", "Trap_15", or "RTS".] Upload the most updated register contents. (96 bytes, if MC68881 is not selected). (96+118 bytes, if MC68881 is selected).
8- Receive new register contents. (96 bytes, if MC68881 is not selected). (96+118 bytes, if MC68881 is selected).	

APPENDIX C: SOURCE CODE OF THE DEBUGGER PROGRAMS

A. SOURCE CODE OF MAIN PROGRAM

i- Source code of download.c

```
/* download.c */


#define   stop10          16384
#define   stop15          ((int) -32768)
#define   stop20          (-16384)
#define   noParity        0
#define   oddParity       4096
#define   evenParity      12288
#define   data7           1024
#define   data8           3072
#define   baud300         380
#define   baud600         189
#define   baud1200        94
#define   baud1800        62
#define   baud2400        46
#define   baud3600        30
#define   baud4800        22
#define   baud7200        14
#define   baud9600        10
#define   activeFlag      0x0001
#define   changeFlag      0x0002
#define   btnState        0x0080
#define   cmdKey          0x0100
#define   shiftKey        0x0200
#define   alphaLock       0x0400
#define   optionKey       0x0800
#define   controlKey      0x1000
#define   charCodeMask    0x000000FFL

enum {    appleID = 1, fileID, optID, widthID, incID };
enum {    quitItem = 1 };
enum {    downItem = 1, SbreakItem, null1Item, regItem , FregItem,
          null2Item, memitem,MemWitem,null3Item,
          Options, DumpItem,null4Item,clearItem, helpItem };
enum {    documentProc, dBoxProc, plainDBox, altDBoxProc, noGrowDocProc,
          rDocProc = 16 };
enum {    fsCurPerm, fsRdPerm, fsWrPerm, fsRdWrPerm, fsRdWrShPerm };
```

```
enum {    fsAtMark, fsFromStart, fsFromLEOF, fsFromMark };
enum {    nullEvent, mouseDown, mouseUp, keyDown, keyUp, autoKey, updateEvt,
          diskEvt, activateEvt };
enum {    inDesk, inMenuBar, inSysWindow, inContent, inDrag, inGrow,
          inGoAway, inZoomIn, inZoomOut };
typedef   unsigned char Str255[256];
typedef   struct { char cumErrs,xOffSent,rdPend,wrPend,ctsHold,
                   xOffHold ;} SerStaRec;
typedef   struct { int menuID; int menuWidth,menuHeight; long menuProc,
                   enableFlags; Str255 menuData; } MenuInfo,* MenuPtr,
                   **MenuHandle;
typedef   char QDByte, *QDPtr, **QDHandle;
typedef   struct { int top,left,bottom,right ; } Rect ;
typedef   struct { QDPtr baseAddr; int rowBytes; Rect bounds; } BitMap;
typedef   struct { int rgnSize; RectrgnBBox; } Region,* RgnPtr,** RgnHandle;
typedef   unsigned char Pattern[8];
typedef   struct { int v,h; } Point ;
typedef   enum   { bold = 1, italic = 2, underline = 4, outline = 8,
                   shadow = 16, condense = 32, extend = 64 } Style;
typedef   struct { QDPtrtextProc; QDPtrlineProc; QDPtrrectProc;
                   QDPtrrRectProc; QDPtrovalProc;QDPtrarcProc;
                   QDPtrpolyProc; QDPtrrgnProc; QDPtrbitsProc;
                   QDPtrcommentProc; QDPtrtxMeasProc; QDPtrgetPicProc;
                   QDPtrputPicProc; } QDProcs,* QDProcsPtr;
typedef   struct GrafPort {
                   int device; BitMap portBits; Rect portRect;
                   RgnHandle visRgn; RgnHandle clipRgn; Pattern bkPat;
                   Pattern fillPat; Point pnLoc; Point pnSize;
                   int pnMode; Pattern pnPat; int pnVis;
                   int txFont; Style txFace; int txMode;
                   int txSize; long spExtra; long fgColor;
                   long bkColor; int colrBit; int patStretch;
                   QDHandle picSave; QDHandle rgnSave; QDHandle
                   QDProcsPtr grafProcs; } GrafPort, * GrafPtr;
typedef   GrafPtrWindowPtr;
typedef   char * Ptr ;
typedef   int (*ProcPtr)() ;
typedef   intOsErr, OSErr;
typedef   unsigned char * StringPtr,** StringHandle ;
typedef   char SignedByte ;
typedef   struct {
                   struct QElem * qLink; int qType,ioTrap; Ptr ioCmdAddr;
                   ProcPtr ioCompletion; OsErr ioResult; StringPtr ioNamePtr
                   int ioVRefNum,ioRefNum; SignedByte ioVersNum,ioPermssn;
                   Ptr ioMisc,ioBuffer; long ioReqCount,ioActCount;
                   int ioPosMode; long ioPosOffset;
                   } ioParam, IOParam ;
typedef   struct EventRecord { int what; long message,when;
                               Point where; int modifiers; }EventRecord;
char      c,instring[255],inbuf[3001],E_bytes[20],fregs[20][8];
```

```
char       Que_buf[2000],*Head,*Tail,*EndQue,*StartQue;
char       DisplaySteps,ReturnTo=2,ErrorFlag=0x00,CameFmGo=0;
char       prnstring[128],prninbuf[3001],prnoutbuf[2500],clrscn;
char       ManSign[8],ExpSign[8],Fbuf[12],RefScrn,OurEvent=0,Reach=1;
int        ByteCount,LastLocCount,scrollsize,LocCount,BreakTimes[5],Clear;
long       registers[24],fcregs[3],Breaks[5],from,to,at;
SerStaRec  SerRec ;
WindowPtr  DisplayWindow;
Rect       windowBounds,myRect,ClrRect;
RgnHandle  myRgn;
ioParam    pbin,pbout,prnbout,prnbin;
MenuHandle appleMenu, fileMenu, optionMenu;

extern     void Error(char *,char *,char *,char *);
extern     void LastScreen(int);
extern     void printhex(long,int);
extern     void DrawChar(char);
extern     GrafPtr   thePort;
pascal     MenuHandle NewMenu();
pascal     WindowPtr NewWindow();
pascal     RgnHandle NewRgn();
extern     BitMapscreenBits;
extern     char *start,*end,Coprocessor,Experienced;
extern     int origin;

main() {
int i,j;

/* Initialize Macintosh Environment */

MaxApplZone();
InitGraf(&thePort);
InitFonts();
FlushEvents(0xFFFF, 0);
InitWindows();
InitMenus();
TEInit();
InitDialogs(0L);
InitCursor();

/* Initialize Menus */

InsertMenu(appleMenu = NewMenu(appleID, "\p\024"), 0);
InsertMenu(fileMenu = NewMenu(fileID, "\pFile"), 0);
InsertMenu(optionMenu = NewMenu(optID, "\pFunctions"), 0);
DrawMenuBar();
AddResMenu(appleMenu, 'DRVR');
AppendMenu(fileMenu, "\pQuit/Q");
AppendMenu(optionMenu, "\pDownload/D;Go.../G;-(;Registers.../R;Floating
           Regs.../F;-(;Memory Display.../M;MemoryWrite.../W;-(;
```

```
          Options/O;Previous Screen/P;-(;Clear Screen/C;Help/H;");

/* Initialize Screen */

myRect.left=4;  windowBounds.left=8;
myRect.top=0;   windowBounds.top=40;
myRect.right  = (windowBounds.right=screenBits.bounds.right-8)-8;
myRect.bottom = (windowBounds.bottom=screenBits.bounds.bottom-8)-4;
                 DisplayWindow = NewWindow(0L, &windowBounds, "\pDisplay",
                 1,noGrowDocProc, -1L, 1, 0);
SetPort(DisplayWindow);
MoveTo(4,myRect.bottom-40);
TextFont(4);
TextSize(scrollsize=9);
SetRectRgn(myRgn=NewRgn(),0,0,0,0);

/* Initialize Printer Port */

prnbin.ioPermssn=fsRdPerm;
prnbin.ioNamePtr= (StringPtr)"\p.BIn";
prnbin.ioVRefNum = 0;
prnbin.ioVersNum= 0;
prnbin.ioMisc = 0L;
prnbin.ioBuffer = prnstring;
PBOpen(&prnbin,0);
prnbout.ioPermssn=fsWrPerm;
prnbout.ioNamePtr = (StringPtr)"\p.BOut";
prnbout.ioVRefNum = 0;
prnbout.ioVersNum= 0;
prnbout.ioMisc = 0L;
PBOpen(&prnbout,0);
prnbout.ioPosMode = prnbin.ioPosMode = fsAtMark;
prnbout.ioPosOffset = prnbin.ioPosOffset = 0;
prnbin.ioRefNum = -8;
prnbout.ioRefNum = -9;
prnbout.ioBuffer = prnoutbuf;
prnbout.ioReqCount = 1;
SerReset(-8,baud9600+noParity+stop20+data8);
SerReset(-9,baud9600+noParity+stop10+data8);
SerSetBuf(-8,prninbuf,3000);

/* Initialize Modem Port */

pbin.ioPermssn=fsRdPerm;
pbin.ioNamePtr= (StringPtr)"\p.AIn";
pbin.ioVRefNum = 0;
pbin.ioVersNum= 0;
pbin.ioMisc = 0L;
pbin.ioBuffer = instring;
PBOpen(&pbin,0);
```

```
pbout.ioPermssn=fsWrPerm;
pbout.ioNamePtr = (StringPtr)"\p.AOut";
pbout.ioVRefNum = 0;
pbout.ioVersNum= 0;
pbout.ioMisc = 0L;
PBOpen(&pbout,0);
pbout.ioPosMode = pbin.ioPosMode = fsAtMark;
pbout.ioPosOffset = pbin.ioPosOffset = 0;
pbin.ioRefNum = -6;
pbout.ioRefNum = -7;
pbout.ioBuffer = &c;
pbout.ioReqCount = 1;
SerReset(-6,baud9600+noParity+stop20+data8);
SerReset(-7,baud9600+noParity+stop20+data8);
SerSetBuf(-6,inbuf,3000);

for(i=0;i<20;i++)
    for(j=0;j<8;j++) fregs[i][j]='0';
for(j=0;j<8;j++) {
    ManSign[j]='+';
    ExpSign[j]='+';
    }
test();
Head=Tail=&Que_buf[0];
EndQue=&Que_buf[1999];
*EndQue=0x00;
StartQue=&Que_buf[0];
Dassy();
for (;;) HandleEvent();
}


/*  HANDLE_EVENT()

    function:
           - This function handles the events.
    arguments:
           - theEvent
    called by:
           - HandleEvent()/download.c
    calls    :
           - HandleMouseDown()/download.c
           - Stop_n_Flush()/download.c
           - doFunction()/menu.c

*/
HandleEvent()
{
EventRecord theEvent;
WindowPtr theWindow;
```

```
int       windowCode,ok,i;
long l;

if(ReturnTo==0) doFunction(2);
if(ReturnTo==1) doFunction(4);
SerStatus (-6,&SerRec) ;
if(SerRec.cumErrs == 64 ) {
  Error("\pError in Transmission!","\p  Try Again...","\p","\p");
  Stop_n_Flush();  }  /* Discard the input while looping outside menu. */
if(!Coprocessor)
  DisableItem(optionMenu,5);
else
  EnableItem(optionMenu,5);
HiliteMenu(0);
SystemTask ();
if (ok = GetNextEvent (0xffff, &theEvent)) {
   switch (theEvent.what) {
case mouseDown: HandleMouseDown(&theEvent);break;
case keyDown: case autoKey:
     if ((theEvent.modifiers & cmdKey) != 0) {
        HandleMenu(MenuKey((char) (theEvent.message & charCodeMask)));
     }
     else
       send(c=theEvent.message & charCodeMask);
     break;
case updateEvt: if(clrscn) {
                  BeginUpdate(DisplayWindow);
                  SetPort(DisplayWindow); EraseRect(&myRect);
                  EndUpdate(DisplayWindow);
                  }
                if(OurEvent) {
                  if(!RefScrn) Clear=2;
                  LastScreen(Clear);
                  OurEvent=0;
                  } break;
case activateEvt: InvalRect(&DisplayWindow->portRect);
                  break;
     }
}
}


/*  HANDLE_MOUSE_DOWN()

    function:
           - This function handles mouse down  operations.
    arguments:
           - theEvent
    called by:
           - HandleEvent()/download.c
```

```
    calls     :
           - None

*/
HandleMouseDown(theEvent) EventRecord   *theEvent;
{
WindowPtr theWindow;
int       windowCode = FindWindow (theEvent->where, &theWindow);

switch (windowCode) {
case inSysWindow: SystemClick (theEvent, theWindow); break;
case inMenuBar: HandleMenu(MenuSelect(theEvent->where)); break;
case inGoAway: if(theWindow==DisplayWindow&&TrackGoAway(DisplayWindow,
               theEvent->where)) HideWindow(DisplayWindow);  break;
 }
}


/*  HANDLE_MENU()

    function:
           - This function handles menu operations.
    arguments:
           - mSelect
    called by:
           - HandleEvent()/download.c
           - HandleMouseDown()/download.c
    calls     :
           - doFunction()/menu.c

*/
HandleMenu (mSelect) long mSelect;
{
int menuID = HiWord(mSelect);
int menuItem = LoWord(mSelect);
Str255 name;
GrafPtr savePort;
long l;

switch (menuID) {
caseappleID: GetPort(&savePort); GetItem(appleMenu, menuItem, name);
             OpenDeskAcc(name); SetPort(savePort); break;
casefileID:
switch (menuItem) {
casequitItem: ExitToShell();
              break;
              } break;
caseoptID: doFunction (menuItem); break;
}
}
```

```
/*  SEND()

    function:
           - This function displays a byte on the Macintosh screen.
    arguments:
           - a
    called by:
           - go()/monitor.c
           - DownLoad()/monitor.c
           - memdisp()/monitor.c
           - wmem()/monitor.c
           - DisAsm()/monitor.c
           - SendRegs()/monitor.c
           - HandleEvent()/download.c
    calls    :
           - None

*/
send(a) char a;
{
long l;

c=a;
PBWrite(&pbout,0);
Delay(1L,&l);
}


/*  SEND_PRN()

    function:
           - This function sends a byte to the serial printer output.
    arguments:
           - a
    called by:
           - DumptoPrn()/Monitor.c
    calls    :
           - None

*/
sendprn(a) char a;
{
long l;

prnoutbuf[0]=a;
PBWrite(&prnbout,0);
Delay(1L,&l);
}
```

```
/*  COPY_REGS()

    function:
           - This function receives and copies the updated Register
             Information which are sent by the ECB.
    arguments:

    called by:
           - go()/monitor.c
           - DownLoad()/monitor.c
    calls    :
           - None

*/
CopyRegs()
{
char instring2[4] ;
int j,m=0,k=0 ;

while (m<24) {
      registers[m]= 0 ;
      for (j=0;j<4;j++) {
          instring2[j]=instring[k]; k++;
          }
      for (j=0;j<4;j++)
          registers[m]=(instring2[j]&0xff)+(registers[m]<<8);
          m++  ;
      }
}


/*  COPY_BRK_CNTS()

    function:
           - This function receives and copies the updated Break Counts
             which are sent by the ECB.
    arguments:

    called by:
           - go()/monitor.c
    calls    :
           - None

*/
CopyBrkCnts()
{
char instring2[2];
int  j,m,k=96,TempLoc;
```

```
for(j=0;j<5;j++) {
   TempLoc=0;
   for(m=0;m<2;m++){
      instring2[m]=instring[k] ; k++ ;
      }
   for(m=0;m<2;m++) TempLoc=(instring2[m]&0xff)+(TempLoc<<8);
   if((TempLoc==0)&&(BreakTimes[j]>=1)) BreakTimes[j]=1;
   else
     BreakTimes[j]=TempLoc;
   }
}


/*  INPUT_BUFFER()

    function:
           - This function checks modem input, and waits until 'hit'
             bytes are received.
    arguments:
           - hit
    called by:
           - DownLoad()/monitor.c
           - go()/monitor.c
           - memdisp()/monitor.c
           - wmem()/monitor.c
           - DisAsm()/monitor.c
    calls    :
           - None

*/
InputBuffer(hit)
int hit ;
{
char c;
int  n;
long l,m;

for(; ;) {
   SerGetBuf(-6,&l);
   if(l>=hit) break ;
   }
if(l!=0) {
  HiliteMenu(fileID);
  if (l>255) l=255; pbin.ioReqCount = l;  PBRead(&pbin,0);
  }
}


/*  CHECK_ERROR()
```

```
    function:
           - This function checks to see whether an error occurred or
             not, during data transmission.
    arguments:

    called by:
           - DownLoad()/monitor.c
           - go()/monitor.c
           - memdisp()/monitor.c
           - wmem()/monitor.c
           - SendRegs()/monitor.c
    calls    :
           - Stop_n_Flush()/download.c
           - Error()/download.c

*/
CheckError()
{
int  n;

for(n=0;n<32767;n++);
SerStatus (-6,&SerRec);
if(SerRec.cumErrs == 64) {
  Error("\pError in Transmission!","\p  Try Again...","\p","\p");
  ErrorFlag=1;
  Stop_n_Flush();
  }
}


/*  STOP_N_FLUSH()

    function:
           - This function stops receiving from modem input, discarding the
             previously received data.
    arguments:

    called by:
           - DownLoad()/monitor.c
           - go()/monitor.c
           - memdisp()/monitor.c
           - dump()/monitor.c
           - doFunction()/menu.c
           - CheckError()/download.c
           - HandleEvent()/download.c
    calls    :
           - None

*/
```

```
Stop_n_Flush()
{
long l;

PBKillIO(&pbin,0);
SerGetBuf(-6,&l);
if(l!=0) {
   pbin.ioReqCount = l;
   PBRead(&pbin,0);
   }
}
```

ii. Source code of menu.c

```
/*  Menu.c  */

#define   NULL            0L
#define   everyEvent      0xFFFF

typedef   int (*ProcPtr)() ;
typedef   struct { inttop,left,bottom,right ; } Rect ;
typedef   char QDByte, *QDPtr, **QDHandle;
typedef   struct { QDPtrbaseAddr; introwBytes; Rectbounds; } BitMap;
typedef   struct { intrgnSize; RectrgnBBox; } Region,* RgnPtr,** RgnHandle;
typedef   struct { intv,h; } Point ;
typedef   unsigned char Pattern[8];
typedef   enum { bold = 1, italic = 2, underline = 4, outline = 8,
                 shadow = 16, condense = 32, extend = 64 } Style;
typedef   struct { QDPtrtextProc,lineProc,rectProc,rRectProc,ovalProc,
                   arcProc,polyProc,rgnProc,bitsProc,commentProc,txMeasProc,
                   getPicProc,putPicProc;} QDProcs,* QDProcsPtr;
typedef   structGrafPort {int device;BitMap portBits;Rect portRect;
                          RgnHandle visRgn,clipRgn; Pattern bkPat,fillPat;
                          Point pnLoc,pnSize; int pnMode; Pattern pnPat;
                          int pnVis,txFont; Style txFace; int txMode,txSize;
                          long spExtra,fgColor,bkColor; int
                          colrBit,patStretch;
                          QDHandle picSave,rgnSave,polySave;
                          QDProcsPtr grafProcs; } GrafPort, * GrafPtr;
typedef   GrafPtr   WindowPtr;
typedef   char ** Handle ;
typedef   unsigned char Str255[256];
typedef   struct { int menuID,menuWidth,menuHeight; Handle menuProc;
                   long enableFlags; Str255menuData;
                   } MenuInfo,* MenuPtr, ** MenuHandle;
typedef   WindowPtr DialogPtr ;
typedef   struct EventRecord { int what; longmessage,when; Pointwhere;
                               int modifiers; } EventRecord;
pascal    DialogPtr GetNewDialog();
enum {    downItem = 1, SbreakItem, null1Item, regItem ,
          FregItem, null2Item, memitem,MemWitem,null3Item,
          Options, DumpItem,null4Item,clearItem, helpItem };
enum {    appleID = 1, fileID, optID, widthID, incID };

extern    char ManSign[8],ExpSign[8],Fbuf[12],OurEvent, DisAsmOutBuf[81];
extern    long StaDisAdr,EndDisAdr;
extern    intBreakTimes[5],Clear;
extern    long registers[24],Breaks[5],fcregs[3],from,to,at;
extern    char fregs[20][8],clrscn,instring[255],DisplaySteps;
extern    char ReturnTo,CameFmGo,Que_buf[2000],*Head,*Tail,*EndQue,RefScrn;
```

```
extern    WindowPtr DisplayWindow;
extern    Rect windowBounds,myRect,ClrRect;
extern    SerRec ;

char      verify,WillGoTo=1,DisAssemble,PrintBuf[2500],Experienced=0;
char      GoToReg,Coprocessor,NotAfterGo=0,Brk_Flag,Hardcopy=0;
char      t[]= "PC=.SR=.USP=.SSP=.ISP=.D0=. D1=. D2=. D3=.D4=. D5=.
          D6=. D7=.A0=. A1=. A2=. A3=.A4=. A5=. A6=. A7=.";
long      value ,tempvalue ;

DialogPtr dp;
extern    void      print(char *);
extern    void      DownLoad(int);
extern    void      dump(void);
extern    void      LastScreen(int);
extern    void      FillQue(int) ;
extern    void      DisAsm();
extern    void      Stop_n_Flush();
extern    void      wmemory(int,int);
extern    void      go(void);
extern    void      DumptoPrn(int);
extern    void      DumptoScreen(int,char *);
extern    void      help(void);
extern    void      ltoa( long , char *, int);
extern    void      itoa( int , char *);
extern    long      atol(char *);
extern    long      atoi(char *);
extern    void      printhex(long,int);
extern    void      prnthex2(long,int,int);
extern    void      CheckHex(int,int);
extern    void      CheckDec(int,int,int);
extern    void      Error(char *,char *,char *,char *);


/*  DO_FUNCTION()

    function :
           - This function provides user interface to the debugger.
             Selection of a particular menu, such as registers menu or Go
             menu, etc., display of that menu, and the user's manipulation
             of the fields in that menu, the update of that menu, etc., thes
             are all provided by DO_FUNCTION().
    arguments:
           - theItem
    called by:
           - HandleEvent()/download.c
           - HandleMenu()//download.c
    calls    :
           - DownLoad()/monitor.c
```

```
            - dump()/monitor.c
            - FillQue()/monitor.c
            - CheckHex()/menu.c
            - CheckDec()/menu.c
            - printhex2()/Monitor.c
            - Stop_n_Flush()/download.c
            - Error()/menu.c
            - ltoa()/monitor.c
            - atol()/monitor.c
            - itoa()/monitor.c
            - atoi()/monitor.c
            - help()/monitor.c
            - wmem()/monitor.c
            - DisAsm()/monitor.c
            - DumptoPrn(i)/monitor.c
            - LastScreen()/monitor.c

*/
doFunction (theItem) int theItem;
{

char       number[21],s[21];
static     char width=1,step=1;
int        i,j,type,change,first,k,mad;
long       l;
Handle     itemh;
EventRecord    myEvent;
Rect       textbox;
clrscn=1;
switch(theItem) {
case downItem:
     DownLoad(0);   break;
case DumpItem:
     LastScreen(1); break;
case regItem:
     dp=GetNewDialog(129,NULL,-1L);
     SetPort(dp);change=1;
     for(i=0;i<24;i++) {
         if(i==19)ltoa(registers[i],number,4);
          else if(i==23)ltoa(registers[i],number,2);
          else ltoa(registers[i],number,8);
          GetDItem(dp,i+2,&type,&itemh,&textbox); SetIText(itemh,number);
          }
     SelIText(dp,2,0,32000);
     do{
     SystemTask();
     GetNextEvent(everyEvent,&myEvent);
     if(change) {
       GetDItem(dp,45,&type,&itemh,&textbox);
       SetCtlValue(itemh,registers[19]&0x010);
```

```
  GetDItem(dp,46,&type,&itemh,&textbox);
  SetCtlValue(itemh,registers[19]&8);
  GetDItem(dp,47,&type,&itemh,&textbox);
  SetCtlValue(itemh,registers[19]&4);
  GetDItem(dp,48,&type,&itemh,&textbox);
  SetCtlValue(itemh,registers[19]&2);
  GetDItem(dp,49,&type,&itemh,&textbox);
  SetCtlValue(itemh,registers[19]&1);
  GetDItem(dp,50,&type,&itemh,&textbox);
  if(((registers[19]&0x3000)==0x2000)&&(Experienced))
    SetCTitle(itemh,"\pSupervisor");
  else
    if(((registers[19]&0x3000)==0x3000)&&(Experienced))
      SetCTitle(itemh,"\pInterrupt");
    else
      SetCTitle(itemh,"\pUser");
  GetDItem(dp,51,&type,&itemh,&textbox);
  i=(registers[19]>>8)&7;
  ltoa((long)i,number,1);
  SetCTitle(itemh,number);
}
if(change==1){
  ltoa(registers[19],number,4);
  GetDItem(dp,21,&type,&itemh,&textbox); SetIText(itemh,number);
  }
ModalDialog(NULL,&theItem);
if(theItem==53) {      /* If ClearAll then Clear Registers D0-A6 */
  ltoa(0L,number,8);
  for(i=0;i<=14;i++)  {
     GetDItem(dp,i+2,&type,&itemh,&textbox);
     SetIText(itemh,number);
     }
  }
if((theItem<26)&&(theItem>1)) CheckHex(theItem,8);
change=0;
if(theItem==21) {
  GetDItem(dp,21,&type,&itemh,&textbox); GetIText(itemh,number);
  registers[19]=atol(number);
  if(((registers[19]&0x00000f00)>>8)>=4) {
    Error("\pInterrupt level >=4       ","\pwill crash the system",
          "\p","\p");
    registers[19]=((registers[19]&0xfffff0ff)|0x00000300);
    ltoa(registers[19],number,4);
    GetDItem(dp,21,&type,&itemh,&textbox);
    SetIText(itemh,number);
    }
    change=2;
   if((!Experienced)&&((registers[19]>>12)!=0)) {
      registers[19]=registers[19]&0xcfff;
      change=1;
```

```
        }
      }
    if(theItem==45) { registers[19] = registers[19] ^ 0x10; change=1;}
    if(theItem==46) {registers[19] = registers[19] ^ 8; change=1;}
    if(theItem==47) {registers[19] = registers[19] ^ 4; change=1;}
    if(theItem==48) {registers[19] = registers[19] ^ 2; change=1;}
    if(theItem==49) {registers[19] = registers[19] ^ 1; change=1;}
    if(theItem==50) {
      if(Experienced){
        i=(registers[19]>>12)&0x07; i=(i+1)%4; i=i<<12;
        registers[19] = (registers[19] & 0xcfff)|i;
        }
    else registers[19] = (registers[19] & 0xcfff);
    change=1;
    }
    if(theItem==51) {
      i=(registers[19]>>8)&7;
      i=(i+1)&7;
      i=(i<<8)&0x0700;j=registers[19]&0xf8ff;
      registers[19] = j|i;
      if(((registers[19]&0x00000f00)>>8)>=4)  {
        Error("\pInterrupt level >=4       ","\pwill crash the system",
              "\p","\p");
        registers[19]=(registers[19]&0xfffff0ff)|0x00000300;
        }
      change=1;
      }
    } while((theItem != 1)&&(theItem != 54));
    for(i=0;i<24;i++) {
       GetDItem(dp,i+2,&type,&itemh,&textbox); GetIText(itemh,number);
       registers[i]=atol(number);
       }
    if((theItem==54)&&(CameFmGo)) {
      ReturnTo=1;
      go();
      }
    else
      if(CameFmGo) ReturnTo=0;
      else  ReturnTo=2;
    DisposDialog(dp);
    SetPort(DisplayWindow);
    Clear=0 ; OurEvent=1;
    break;
case FregItem:
    dp=GetNewDialog(133,NULL,-1L);
    SetPort(dp);
    SelIText(dp,2,0,32000); change=1;
    for(i=0;i<8;i++) {
       number[0]=17;
       for(j=0;j<17;j++) number[j+1]=fregs[j][i];
```

```
    GetDItem(dp,2*(1+i),&type,&itemh,&textbox); SetIText(itemh,number);
    number[0]=3;
    for(j=0;j<3;j++) number[j+1]=fregs[j+17][i];
    GetDItem(dp,2*(1+i)+1,&type,&itemh,&textbox); SetIText(itemh,number);
    number[0]=1; number[1]=ManSign[i];
    GetDItem(dp,2*(17+i),&type,&itemh,&textbox); SetIText(itemh,number);
    number[0]=1; number[1]=ExpSign[i];
    GetDItem(dp,2*(17+i)+1,&type,&itemh,&textbox); SetIText(itemh,number);
    }
for(i=0;i<3;i++){
    ltoa(fcregs[i],number,8);
    GetDItem(dp,i+18,&type,&itemh,&textbox); SetIText(itemh,number);
    }
do{
SystemTask();
GetNextEvent(everyEvent,&myEvent);
if(change) {
  GetDItem(dp,52,&type,&itemh,&textbox);
  SetCtlValue(itemh,((fcregs[0]>>1)&0x00004000));
  GetDItem(dp,53,&type,&itemh,&textbox);
  SetCtlValue(itemh,fcregs[0]&0x004000);
  GetDItem(dp,54,&type,&itemh,&textbox);
  SetCtlValue(itemh,fcregs[0]&0x002000);
  GetDItem(dp,55,&type,&itemh,&textbox);
  SetCtlValue(itemh,fcregs[0]&0x001000);
  GetDItem(dp,56,&type,&itemh,&textbox);
  SetCtlValue(itemh,fcregs[0]&0x000800);
  GetDItem(dp,57,&type,&itemh,&textbox);
  SetCtlValue(itemh,fcregs[0]&0x000400);
  GetDItem(dp,58,&type,&itemh,&textbox);
  SetCtlValue(itemh,fcregs[0]&0x000200);
  GetDItem(dp,59,&type,&itemh,&textbox);
  SetCtlValue(itemh,fcregs[0]&0x000100);
  GetDItem(dp,60,&type,&itemh,&textbox);
  SetCtlValue(itemh,(fcregs[1]>>24)&0x08);
  GetDItem(dp,61,&type,&itemh,&textbox);
  SetCtlValue(itemh,(fcregs[1]>>24)&0x04);
  GetDItem(dp,62,&type,&itemh,&textbox);
  SetCtlValue(itemh,(fcregs[1]>>24)&0x02);
  GetDItem(dp,63,&type,&itemh,&textbox);
  SetCtlValue(itemh,(fcregs[1]>>24)&0x01);
  }
if (change==1){
    ltoa(fcregs[0],number,4);
    GetDItem(dp,18,&type,&itemh,&textbox); SetIText(itemh,number);
    ltoa(fcregs[1],number,8);
    GetDItem(dp,19,&type,&itemh,&textbox); SetIText(itemh,number);
    ltoa(fcregs[2],number,8);
    GetDItem(dp,20,&type,&itemh,&textbox); SetIText(itemh,number);
    }
```

```
    ModalDialog(NULL,&theItem);
    if(((theItem<17)&&(theItem>1))&&((theItem % 2)==0)) CheckHex(theItem,17);
      if(((theItem<18)&&(theItem>2))&&(((theItem+1)%2)==0))
    CheckHex(theItem,3);
    if((theItem<50)&&(theItem>33)) CheckDec(theItem,1,1);
    change=0;
    if(theItem==18) {
      GetDItem(dp,18,&type,&itemh,&textbox); GetIText(itemh,number);
      fcregs[0]=atol(number); change=2;
      }
    if(theItem==19) {
      GetDItem(dp,19,&type,&itemh,&textbox); GetIText(itemh,number);
      fcregs[1]=atol(number); change=2;
    }
    if(theItem==53) { fcregs[0] = fcregs[0] ^ 0x004000;   change=1; }
    if(theItem==54) { fcregs[0] = fcregs[0] ^ 0x002000;   change=1; }
    if(theItem==55) { fcregs[0] = fcregs[0] ^ 0x001000;   change=1; }
    if(theItem==56) { fcregs[0] = fcregs[0] ^ 0x000800;   change=1; }
    if(theItem==57) { fcregs[0] = fcregs[0] ^ 0x000400;   change=1; }
    if(theItem==58) { fcregs[0] = fcregs[0] ^ 0x000200;   change=1; }
    if(theItem==52) { fcregs[0] = fcregs[0] ^ 0x008000;   change=1; }
    if(theItem==59) { fcregs[0] = fcregs[0] ^ 0x000100;   change=1; }
    if(theItem==60) { fcregs[1] = fcregs[1] ^ 0x08000000; change=1; }
    if(theItem==61) { fcregs[1] = fcregs[1] ^ 0x04000000; change=1; }
    if(theItem==62) { fcregs[1] = fcregs[1] ^ 0x02000000; change=1; }
    if(theItem==63) { fcregs[1] = fcregs[1] ^ 0x01000000; change=1; }

    } while(theItem != 1);
    for(i=0;i<8;i++){
       GetDItem(dp,2*(1+i),&type,&itemh,&textbox); GetIText(itemh,number);
       for(j=0;j<17;j++) fregs[j][i]=number[j+1];
       GetDItem(dp,2*(1+i)+1,&type,&itemh,&textbox); GetIText(itemh,number);
       for(j=0;j<3;j++) fregs[j+17][i]=number[j+1];
       }
    for(i=0;i<3;i++){
       GetDItem(dp,i+18,&type,&itemh,&textbox); GetIText(itemh,number);
       fcregs[i]=atol(number);
       }
    for(i=0;i<16;i+=2){
       GetDItem(dp,34+i,&type,&itemh,&textbox); GetIText(itemh,number);
       ManSign[i/2]=number[1];
       GetDItem(dp,34+i+1,&type,&itemh,&textbox); GetIText(itemh,number);
       ExpSign[i/2]=number[1];
       }
    DisposDialog(dp); SetPort(DisplayWindow);
    Clear=0; OurEvent=1;
    break;
case memitem:
    first=1;
    dp=GetNewDialog(130,NULL,-1L); SetPort(dp);change=0;
```

```
ltoa(from,number,8);
GetDItem(dp,2,&type,&itemh,&textbox); SetIText(itemh,number);
ltoa(to,number,8);
GetDItem(dp,3,&type,&itemh,&textbox); SetIText(itemh,number);
ltoa(to-from,number,8);
GetDItem(dp,4,&type,&itemh,&textbox); SetIText(itemh,number);
GetDItem(dp,9,&type,&itemh,&textbox); SetCtlValue(itemh,DisAssemble);
SelIText(dp,2,0,32000);
do{
if(change==1){
  ltoa(to-from,number,8);
  GetDItem(dp,4,&type,&itemh,&textbox); SetIText(itemh,number);
  }
if(change==2){
  ltoa(to,number,8);
  GetDItem(dp,3,&type,&itemh,&textbox); SetIText(itemh,number);
  }
SystemTask();
GetNextEvent(everyEvent,&myEvent);
ModalDialog(NULL,&theItem);
change=0;
if(theItem==2) {
  GetDItem(dp,2,&type,&itemh,&textbox); GetIText(itemh,number);
  from=atol(number);change=1;
  }
if(theItem==3) {
GetDItem(dp,3,&type,&itemh,&textbox); GetIText(itemh,number);
to=atol(number);change=1;
}
if(theItem==4) {
  GetDItem(dp,4,&type,&itemh,&textbox); GetIText(itemh,number);
  to=atol(number)+from;change=2;
  }
if((first&&DisAssemble)||(theItem==9)) {
  GetDItem(dp,9,&type,&itemh,&textbox);
  SetCtlValue(itemh,DisAssemble=(theItem==9)? !DisAssemble:DisAssemble)
  }
first=0;
} while((theItem != 1)&&(theItem != 8));
DisposDialog(dp); SetPort(DisplayWindow);
clrscn=0;
if((to-from>=500)&&(theItem==1)&&(!DisAssemble)) {
  to=from+500;
  Error("\pCannot dump more than ","\p 500 bytes at        a
        time.","\p","\p");
  }
if((to<from)&&(theItem==1)) {
  ltoa(from,s,8);ltoa(to,number,8);
  Error("\pCannot dump range ",s,"\p to ",number);
  }
```

```
    else if(theItem==1) {
           if(DisAssemble) {
             StaDisAdr=from; EndDisAdr=to;
             NotAfterGo=1; DisAsm();
             }
             else {
               NotAfterGo=0;  dump();
               }
             }
            if(RefScrn) LastScreen(0);
         Clear=0; OurEvent=0;
         break;
case MemWitem:
     dp=GetNewDialog(132,NULL,-1L); SetPort(dp);change=0;
     ltoa(at,number,8);
     GetDItem(dp,2,&type,&itemh,&textbox); SetIText(itemh,number);
     ltoa(value,number,width*2);
     GetDItem(dp,3,&type,&itemh,&textbox); SetIText(itemh,number);
     first=1;theItem=0; SelIText(dp,2,0,32000);
     do{
     SystemTask();
     GetNextEvent(everyEvent,&myEvent);
     if((first&&width==1)||(theItem==6)){
       GetDItem(dp,6,&type,&itemh,&textbox);  SetCtlValue(itemh,1);
       GetDItem(dp,7,&type,&itemh,&textbox);  SetCtlValue(itemh,0);
       GetDItem(dp,8,&type,&itemh,&textbox);  SetCtlValue(itemh,0);width=1;
       }
     if((first&&width==2)||(theItem==7)){
       GetDItem(dp,6,&type,&itemh,&textbox);  SetCtlValue(itemh,0);
       GetDItem(dp,7,&type,&itemh,&textbox);  SetCtlValue(itemh,1);
       GetDItem(dp,8,&type,&itemh,&textbox);  SetCtlValue(itemh,0);width=2;
       }
     if((first&&width==4)||(theItem==8)){
       GetDItem(dp,6,&type,&itemh,&textbox);  SetCtlValue(itemh,0);
       GetDItem(dp,7,&type,&itemh,&textbox);  SetCtlValue(itemh,0);
       GetDItem(dp,8,&type,&itemh,&textbox);  SetCtlValue(itemh,1);width=4;
       }
     if((first&&step==1)||(theItem==9)){
       GetDItem(dp,9,&type,&itemh,&textbox);  SetCtlValue(itemh,1);
       GetDItem(dp,10,&type,&itemh,&textbox); SetCtlValue(itemh,0);
       GetDItem(dp,11,&type,&itemh,&textbox); SetCtlValue(itemh,0);step=1;
       }
     if((first&&step==0)||(theItem==10)){
       GetDItem(dp,9,&type,&itemh,&textbox);  SetCtlValue(itemh,0);
       GetDItem(dp,10,&type,&itemh,&textbox); SetCtlValue(itemh,1);
       GetDItem(dp,11,&type,&itemh,&textbox); SetCtlValue(itemh,0);step=0;
       }
     if((first&&step==-1)||(theItem==11)){
       GetDItem(dp,9,&type,&itemh,&textbox);  SetCtlValue(itemh,0);
       GetDItem(dp,10,&type,&itemh,&textbox); SetCtlValue(itemh,0);
```

```
      GetDItem(dp,11,&type,&itemh,&textbox); SetCtlValue(itemh,1);step=-1;
      }
    if((first&&verify)||(theItem==12)){
      GetDItem(dp,12,&type,&itemh,&textbox);
      SetCtlValue(itemh,verify=(theItem==12) ? !verify: verify);
      }
    first=0;
    ModalDialog(NULL,&theItem);
    if(theItem==3) change=1;
      if((theItem==2) && change){
        GetDItem(dp,3,&type,&itemh,&textbox); GetIText(itemh,number);
        value=atol(number);
        GetDItem(dp,2,&type,&itemh,&textbox); GetIText(itemh,number);
        at=atol(number);
        wmemory(step,width);
        ltoa(at,number,8);
        GetDItem(dp,2,&type,&itemh,&textbox);
        SetIText(itemh,number); SelIText(dp,3,0,80);
        ltoa(value,number,width*2);
        GetDItem(dp,3,&type,&itemh,&textbox);
        SetIText(itemh,number); SelIText(dp,3,0,80);
        }
    } while(theItem != 1);
    GetDItem(dp,2,&type,&itemh,&textbox); GetIText(itemh,number);
    at=atol(number);
    DisposDialog(dp); SetPort(DisplayWindow);
    Clear=0; OurEvent=1;
    break;
case SbreakItem:
    dp=GetNewDialog(131,NULL,-1L); SetPort(dp);
    change=0; Brk_Flag=0x00;
    for(i=0;i<5;i++){
       ltoa(Breaks[i],number,8);
       GetDItem(dp,i+2,&type,&itemh,&textbox); SetIText(itemh,number);
       itoa(BreakTimes[i],number);
       GetDItem(dp,i+14,&type,&itemh,&textbox); SetIText(itemh,number);
       }
    ltoa(registers[18],number,8);
    SelIText(dp,10,0,32000);
    GetDItem(dp,10,&type,&itemh,&textbox); SetIText(itemh,number);
    first=change=1;theItem=0;
    do{
    SystemTask();
    GetNextEvent(everyEvent,&myEvent);
    if((first&&WillGoTo==1)||(theItem==8)){
      GetDItem(dp,8,&type,&itemh,&textbox); SetCtlValue(itemh,WillGoTo=1);
      GetDItem(dp,9,&type,&itemh,&textbox); SetCtlValue(itemh,0);
      }
    if((first&&WillGoTo==0)||(theItem==9)){
      GetDItem(dp,8,&type,&itemh,&textbox); SetCtlValue(itemh,WillGoTo=0);
```

```
  GetDItem(dp,9,&type,&itemh,&textbox); SetCtlValue(itemh,1);
  }
for(i=0;i<5;i++){
    GetDItem(dp,i+2,&type,&itemh,&textbox); GetIText(itemh,number);
    tempvalue=atol(number);
    if ((Breaks[i])==tempvalue) {
       GetDItem(dp,i+14,&type,&itemh,&textbox); GetIText(itemh,number);
       BreakTimes[i]=atoi(number);
       }
    else {
      Breaks[i]=tempvalue;
      if(Breaks[i]!=0x00) {
        BreakTimes[i]=1;
        itoa(BreakTimes[i],number);
        GetDItem(dp,i+14,&type,&itemh,&textbox); SetIText(itemh,number);
        }
      }
    if(Breaks[i]==0x00) {
      BreakTimes[i]=0;
      itoa(BreakTimes[i],number);
      GetDItem(dp,i+14,&type,&itemh,&textbox); SetIText(itemh,number);
      }
if(change){
  GetDItem(dp,13,&type,&itemh,&textbox);
  if((registers[19]&0xc000)==0x8000) SetCTitle(itemh,"\pTrace All");
  else
    if((registers[19]&0xc000)==0x4000) SetCTitle(itemh,"\pTrace Branch");
  else SetCTitle(itemh,"\pNo Trace");
  for(i=0;i<5;i++){
     ltoa(Breaks[i],number,8);
     GetDItem(dp,i+2,&type,&itemh,&textbox); SetIText(itemh,number);
     itoa(BreakTimes[i],number);
     GetDItem(dp,i+14,&type,&itemh,&textbox); SetIText(itemh,number);
     }
  GetDItem(dp,19,&type,&itemh,&textbox); SetCtlValue(itemh,DisplaySteps);
  GetDItem(dp,21,&type,&itemh,&textbox);
  if((ReturnTo==1)||((ReturnTo==0)&&(GoToReg))) {
    SetCTitle(itemh,"\pRegister Menu");
    ReturnTo=1; GoToReg=0;
    }
  else if(ReturnTo==0) SetCTitle(itemh,"\pGo Menu");
  else SetCTitle(itemh,"\pNo Menu");
  change=0;
  }
  ModalDialog(NULL,&theItem);
  if(((theItem>=2)&&(theItem<=6))||(theItem==10)) CheckHex(theItem,8);
  if((theItem>=14)&&(theItem<=18)) CheckDec(theItem,4,9);
  if(theItem==7){
    for(i=0;i<5;i++){
       ltoa(Breaks[i]=0,number,8);
```

```
        GetDItem(dp,i+2,&type,&itemh,&textbox); SetIText(itemh,number);
        itoa(BreakTimes[i]=0,number);
        GetDItem(dp,i+14,&type,&itemh,&textbox); SetIText(itemh,number);
        }
     change=1;
     }
if(theItem==19) {
  DisplaySteps= ! DisplaySteps;
  change=1;
  }
if(theItem==21) { ReturnTo=(ReturnTo+1) % 3; change=1; }
if(theItem==13){
  registers[19]=(registers[19]&0x3fff)|((((registers[19]>>14)+1)%3)<<14
  change=1; first=0;
  }
} while((theItem != 1)&&(theItem != 11));
for(i=0;i<5;i++){
   GetDItem(dp,i+2,&type,&itemh,&textbox); GetIText(itemh,number);
   Breaks[i]=atol(number);
   GetDItem(dp,i+14,&type,&itemh,&textbox); GetIText(itemh,number);
   BreakTimes[i]=atoi(number);
   }
GetDItem(dp,10,&type,&itemh,&textbox); GetIText(itemh,number);
registers[18]=atol(number);
if(theItem==1) {
  if(ReturnTo==1) CameFmGo=GoToReg=1;
  else if(ReturnTo==0) GoToReg=0;
  else CameFmGo=GoToReg=0;
  if(registers[18]<0x1000) {
    Error("\pIllegal Go Address...","\p  Must be over $1000.","\p","\p
    registers[18]=0x00; Brk_Flag=1;
    }
  for(i=0;i<5;i++)
     if((Breaks[i]< 0x1000)&&(BreakTimes[i]!=0)) {
       ltoa((long)(i+1),s,2); Brk_Flag=1;
       Error("\pIllegal Breakpoint #",s,"\p  Must be over $1000.","\p"
       Breaks[i]=0x0000; BreakTimes[i]=0;
       }
  }
if((theItem==1)&&(!Brk_Flag )) go();
  if(theItem==11) {
    ReturnTo=2;
    CameFmGo=GoToReg=0;
    }
DisposDialog(dp); SetPort(DisplayWindow); clrscn=0;
if((theItem==1)&&(!Brk_Flag )) {
  if((ReturnTo!=0)&&(ReturnTo!=1)) {
    for(i=0,j=0;j<99;j++) {
       if(t[j]=='.') {
         for(k=0;k<9;k++) { PrintBuf[i]=' ';i++;}
```

```
            if((j==3)||(j==22)||(j==41)||(j==60)||(j==79)||(j==98)) {
            PrintBuf[i]=0x0d; i++; PrintBuf[i]=0x0a;i++;
            }
        }
    else {  PrintBuf[i]=t[j]; i++; }
    }
    prnthex2(registers[18],8,3);
    prnthex2(registers[19],4,17);
    prnthex2(registers[15],8,30);
    prnthex2(registers[16],8,43);
    prnthex2(registers[17],8,56);
    for(i=70,j=0;i<281;i+=13,j++) {
       if((i==122)||(i==175)||(i==228)) i+=1;
       prnthex2(registers[j],8,i);
       }
    StaDisAdr=EndDisAdr=registers[18];
    if(!RefScrn) DumptoScreen(i,&PrintBuf[0]);
    Stop_n_Flush();
    DisAsm();
    prnthex2(StaDisAdr,8,i); i+=8;
    PrintBuf[i]=0x20;i++;
    for(j=i,k=1;k<81;k++,j++) PrintBuf[j]=DisAsmOutBuf[k]; i=j;
    PrintBuf[i]=0x0d;i++; PrintBuf[i]=0x0a; i++;
    PrintBuf[i]=0x0d;i++; PrintBuf[i]=0x0a;
    FillQue(i-1);
    if(Hardcopy==1)  DumptoPrn(i);
    }
    }
    if(ReturnTo!=2) Clear=2;
    else Clear=0;
    OurEvent=1;
    Stop_n_Flush();
    break;
case clearItem:
    EraseRect(&myRect); Clear=2; break;
case helpItem:
    EraseRect(&myRect); clrscn=0;
    help(); break;
case Options:
    dp=GetNewDialog(135,NULL,-1L); SetPort(dp);
    GetDItem(dp,5,&type,&itemh,&textbox); SetCtlValue(itemh,Experienced);
    GetDItem(dp,4,&type,&itemh,&textbox); SetCtlValue(itemh,RefScrn);
    GetDItem(dp,3,&type,&itemh,&textbox); SetCtlValue(itemh,Hardcopy);
    GetDItem(dp,2,&type,&itemh,&textbox); SetCtlValue(itemh,Coprocessor);
    first=1; theItem=0;
    do{
    SystemTask();
    GetNextEvent(everyEvent,&myEvent);
    if((first&&Experienced)||(theItem==5)) {
      GetDItem(dp,5,&type,&itemh,&textbox);
```

```
        SetCtlValue(itemh,Experienced=(theItem==5)? !Experienced:Experienced)
        }
      if((first&&RefScrn)||(theItem==4)) {
        GetDItem(dp,4,&type,&itemh,&textbox);
        SetCtlValue(itemh,RefScrn= (theItem==4) ? !RefScrn: RefScrn);
        }
      if((first&&Hardcopy)||(theItem==3)) {
        GetDItem(dp,3,&type,&itemh,&textbox);
        SetCtlValue(itemh,Hardcopy= (theItem==3) ? !Hardcopy: Hardcopy);
        }
      if((first&&Coprocessor)||(theItem==2)) {
        GetDItem(dp,2,&type,&itemh,&textbox);
        SetCtlValue(itemh,Coprocessor=(theItem==2)? !Coprocessor:Coprocessor)
        }
      first=0;
      ModalDialog(NULL,&theItem);
      } while(theItem != 1);
      DisposDialog(dp); SetPort(DisplayWindow);
      Clear=0; OurEvent=1;
      }
      return(1);
}


/*  CHECK_HEX()

    function :
          - This function checks to see if its argument is a valid
            hexadecimal number or not.  If the number is not valid,
            an error message is displayed, and that entry is cleared.
    arguments:
          - theItem,n
    called by:
          - doFunction()/menu.c
    calls    :
          - Error()/menu.c
          - ltoa()/monitor.c

*/
void CheckHex(theItem,n) int theItem,n;{

char number[21];
int i,j,type;
Handle itemh;
Rect textbox;
GetDItem(dp,theItem,&type,&itemh,&textbox); GetIText(itemh,number);
if(number[0]>n) {
  Error( "\pToo Many Digits in: ", number,"\p","\p");
  ltoa(0L,number,n);
  GetDItem(dp,theItem,&type,&itemh,&textbox); SetIText(itemh,number);
```

```
  }
for(i=1,j=0;i<number[0];i++)
   j|=((number[i]<'0')||((number[i]>'9')&&(number[i]<'A'))
   ||((number[i]>'F')&&(number[i]<'a'))||(number[i]>'f'));
if(j) {
  Error( "\pIllegal Hexadecimal Character in: ",number,"\p","\p");
  ltoa(0L,number,n);
  GetDItem(dp,theItem,&type,&itemh,&textbox); SetIText(itemh,number);
  }
}


/*  CHECK_DEC()

    function :
           - This function checks to see if its argument is a valid
             Decimal number or not.  If the number is not valid, an
             error message is displayed, and that entry is cleared.
    arguments:
           - theItem, n, n2
    called by:
           - doFunction()/menu.c
    calls    :
           - Error()/menu.c
           - ltoa()/monitor.c

*/
void CheckDec(theItem,n,n2) int theItem,n,n2;{

char number[21];
int i,j,type;
Handle itemh;
Rect textbox;
GetDItem(dp,theItem,&type,&itemh,&textbox); GetIText(itemh,number);
if(number[0]>n) {
  Error( "\pToo Many Digits in: ", number,"\p","\p");
  ltoa(0L,number,n);
  GetDItem(dp,theItem,&type,&itemh,&textbox); SetIText(itemh,number);
  }
for(i=1,j=0;i<number[0];i++) j|=((number[i]<'0')||(number[i]>'n2'));
if(j) {
  Error( "\pIllegal Decimal Character in: ",number,"\p","\p");
  ltoa(0L,number,n);
  GetDItem(dp,theItem,&type,&itemh,&textbox); SetIText(itemh,number);
  }
}


/*  ERROR()
```

```
    function :
           - This function displays an Error Message on the screen.
             The displayed message is a Pascal string, which is passed
             as a parameter.
    arguments:
           - s, f, l, z
    called by:
           - doFunction()/menu.c
           - CheckDec()/menu.c
           - CheckHex()/menu.c
           - go()/Monitor.c
           - wmem()/Monitor.c
           - CopyFloat()/Monitor.c
           - memdisp()/Monitor.c
           - HandleEvent()/download.c
           - CheckError()/download.c
    calls    :
           - None

*/
void Error( s, f ,l ,z) char *s, *f, *l, *z; {

ParamText(s, f, l, z);
Alert(256, 0L );
}
```

iii. Source code of monitor.c

```
/* Monitor.c */

typedef struct { int v,h; } Point ;
typedef struct { int top,left,bottom,right ; } Rect ;
typedef struct { int rgnSize; RectrgnBBox; } Region,* RgnPtr,** RgnHandle;
extern  char *start,*end,*Head,*Tail,*EndQue,*StartQue;
extern  char fregs[20][8],clrscn,verify,DisplaySteps,ReturnTo;
extern  char c,instring[255],inbuf[3001],E_bytes[20],fregs[20][8];
extern  char Hardcopy,Coprocessor,WillGoTo,RefScrn,Reach,clrscn;
extern  char ErrorFlag,NotAfterGo,PrintBuf[2500],DisAssemble;
extern  char ManSign[8],ExpSign[8],Fbuf[12],Que_buf[2000];
extern  int origin,BreakTimes[5];
extern  long registers[24],Breaks[5],fcregs[3],from,to,at,value;
extern  void send(char);
extern  void sendprn(char);
extern  void Error(char *,char *,char *,char *);
extern  void InputBuffer();
extern  void CheckError();
extern  void CopyRegs();
extern  void CopyBrkCnts();
extern  void Stop_n_Flush();
extern  Rect windowBounds,myRect,ClrRect;
extern  int ByteCount,LastLocCount,scrollsize,LocCount;
extern  RgnHandle myRgn;
char    tbuf[]= "          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F";
char    DisAsmOutBuf[81],AbortEvent=0,AbortCount=0;
int     z,line_count,numofchars;
long    StaDisAdr,EndDisAdr,*HISPC,*SubrAdr;


/*  DUMP()
    function:
          - This function performs the 'Memory Display' operation.
            The two global variables (from, to) are set by the user.
            Size= to-from, bytes of memory are displayed, starting from
            the address 'from'.
            Also, user has the 'Disassemble' option. In this case, the
            memory contents are first disassembled, and then displayed
            on the screen.
    arguments:

    called by:
          - doFunction()/menu.c
    calls    :
          - Stop_n_Flush()/download.c
            Draw_x()/monitor.c
```

```
                memdisp()/monitor.c
                prnthex3()/monitor.c
                FillQue()/monitor.c
                DumptoScreen()/monitor.c
                DumptoPrn()/monitor.c

*/
dump()
{
char DisBuf[16];
int i,inbytes,dis1,size,residual=0,inex;
long from2,baseadr;

Stop_n_Flush();
line_count=0;
for(z=0;z<56;z++) PrintBuf[z]=tbuf[z];
PrintBuf[z]=0x0d;
PrintBuf[z+1]=0x0a; z+=2;
size=((int)(to-from))+1;
numofchars=size;
from2=from;
if(size > 16) {
  if((from2&0x0000000f)!=0) {
     inbytes=16-(int)(from2&0x0000000f);
     baseadr=from2&0xfffffff0 ;
     }
  else {
   inbytes=16;
   baseadr=from2;
   }
  prnthex3(from2&0x0fffffff0L,8,z);
  for(i=0,dis1=0;i<(16-inbytes);dis1++,i++) {
     Draw_x(z);
     DisBuf[dis1]='.';
     }
  memdisp(from2,inbytes);
  for(i=0;i<inbytes;i++,dis1++) {
     if((instring[i]>=0x20)&&(instring[i]<=0x7e))
       DisBuf[dis1]=instring[i];
     else
       DisBuf[dis1]='.';
     }
  PrintBuf[z]=' '; z++;
  for(i=0;i<16;i++,z++) PrintBuf[z]=DisBuf[i];
  PrintBuf[z]=0x0d;
  PrintBuf[z+1]=0x0a; z+=2;
  numofchars-=inbytes ;
  baseadr+=16;
  if((line_count==22)&&(numofchars>=16)) {
    line_count=0;
```

```
    for(inex=z;inex<z+56;inex++) PrintBuf[inex]=tbuf[inex-z];
    PrintBuf[inex]=0x0d;
    PrintBuf[inex+1]=0x0a;
    inex+=2; z=inex;
    }
while(numofchars>16) {
     prnthex3(baseadr,8,z);
     memdisp(baseadr,inbytes=16);
     for(dis1=0;dis1<inbytes;dis1++)  {
        if((instring[dis1]>=0x20)&&(instring[dis1]<=0x7e))
          DisBuf[dis1]=instring[dis1];
        else
          DisBuf[dis1]='.';
        }
     PrintBuf[z]=' '; z++;
     for(i=0;i<16;i++,z++) PrintBuf[z]=DisBuf[i];
     PrintBuf[z]=0x0d;
     PrintBuf[z+1]=0x0a; z+=2;
     numofchars-=inbytes ;
     baseadr+=16;
     if((line_count==22)&&(numofchars>=16)) {
       line_count=0;
       PrintBuf[z]=0x0d; PrintBuf[z+1]=0x0a; z+=2;
       for(inex=z;inex<z+56;inex++) PrintBuf[inex]=tbuf[inex-z];
       PrintBuf[inex]=0x0d;
       PrintBuf[inex+1]=0x0a;
       inex+=2; z=inex;
       }
     }
prnthex3(baseadr,8,z);
memdisp(baseadr,inbytes=numofchars);
for(i=0;i<(16-inbytes);i++) Draw_x(z);
for(i=0;i<inbytes;i++)  {
   if((instring[i]>=0x20)&&(instring[i]<=0x7e))
     DisBuf[i]=instring[i];
   else
     DisBuf[i]='.';
   }
PrintBuf[z]=' '; z++;
for(i=0;i<inbytes;i++,z++) PrintBuf[z]=DisBuf[i];
PrintBuf[z]=0x0d;
PrintBuf[z+1]=0x0a; z+=2;
}
else {
  prnthex3(from2&0x0fffffff0L,8,z);
  if((from2&0x0000000f)!=0)
    for(i=0,dis1=0;i<((int)(from2&0x0000000f));dis1++,i++) {
       Draw_x(z);
       residual++;
       DisBuf[dis1]='.';
```

```
        }
    else dis1=0;
    if(size<=(16-residual)) {
      memdisp(from2,size);
      for(i=0;i<size;dis1++,i++)  {
         if((instring[i]>=0x20)&&(instring[i]<=0x7e))
           DisBuf[dis1]=instring[i];
         else
           DisBuf[dis1]='.';
         }
      for(i=0;i<(16-(size+residual));i++) Draw_x(z);
          PrintBuf[z]=' '; z++;
          for(i=0;i<(residual+size);i++,z++) PrintBuf[z]=DisBuf[i];
          PrintBuf[z]=0x0d; PrintBuf[z+1]=0x0a; z+=2;
          }
        else {
          memdisp(from2,(16-residual));
          for(i=0;i<(16-residual);dis1++,i++) {
             if((instring[i]>=0x20)&&(instring[i]<=0x7e))
               DisBuf[dis1]=instring[i];
             else
               DisBuf[dis1]='.';
             }
          PrintBuf[z]=' '; z++;
          for(i=0;i<16;i++,z++) PrintBuf[z]=DisBuf[i];
          PrintBuf[z]=0x0d; PrintBuf[z+1]=0x0a; z+=2;
          baseadr=from2-residual+16;
          prnthex3(baseadr,8,z);
          memdisp(baseadr,(size-(16-residual)));
          for(i=0;i<(16-(size-16+residual));i++) Draw_x(z);
          for(i=0,dis1=0;i<(16-(size-16+residual));dis1++,i++) {
             if((instring[i]>=0x20)&&(instring[i]<=0x7e))
               DisBuf[dis1]=instring[i];
             else
               DisBuf[dis1]='.';
             }
          PrintBuf[z]=' '; z++;
          for(i=0;i<(size-(16-residual));i++,z++) PrintBuf[z]=DisBuf[i];
          PrintBuf[z]=0x0d; PrintBuf[z+1]=0x0a; z+=2;
  }
}
PrintBuf[z]=0x0d; PrintBuf[z+1]=0x0a;
PrintBuf[z+2]=0x0d; PrintBuf[z+3]=0x0a; z+=3;
FillQue(z-1);
if(!RefScrn)  DumptoScreen(z,&PrintBuf[0]);
if(Hardcopy)  DumptoPrn(z);
AbortEvent=0; AbortCount=0;

}
```

```
/*  MEM_DISP()

    function:
           - This function helps 'dump()' in performing the 'Memory Display'
             operation.
             Maximum sixteen bytes can be handled by this function.
    arguments:
           - staradr, bytecount
    called by:
           - dump()/monitor.c
    calls    :
           - send()/download.c
             Stop_n_Flush()/download.c
             Error()/download.c
             CheckError()/download.c

*/
memdisp(staradr,bytecount)
int  bytecount ;
long staradr;
{
char c,md_code=0x04 ;
int i,chksum;
long l;

send(md_code);
for(i=0;i<=400;i++);
for(i=24;i>=0;i-=8) send(c=(char)(staradr>>i));
SerGetBuf(-6,&l);
if(l>0) {
  numofchars=15;
  AbortEvent=1;
  }
send(c=(char)((bytecount)>>0));
CheckError();
if(!ErrorFlag) {
  InputBuffer(bytecount+1);
  for(i=0;i<bytecount;i++) prnthex3(((long)(instring[i])),2,z);
   for(i=0;i<bytecount;i++)  {          /* calculate checksum */
      if(i==0) chksum =(instring[i] & 0xff) & 0xff ;
      if(i>0 ) chksum^=(instring[i] & 0xff) & 0xff ;
      }
   if ((chksum!=((instring[i] & 0xff) & 0xff))&&(!AbortEvent))
      Error("\pChecksum error.Restart","\p","\p","\p");
   if((AbortEvent)&&(AbortCount==0)) {
     Error("\pBoard Aborted...","\p","\p","\p");
     AbortCount++;
     }
}
```

```
ErrorFlag=0x00;
line_count++;
Stop_n_Flush();

}


/*  W_MEMORY()

    function:
           - This function performs the 'Memory Modify' operation. 'Verify'
             option is also available to the user. In this case, a write is
             done to, and following this, a read from that memory location
             is perfomed. Then, the two are compared.
    arguments:
           - step, width
    called by:
           - doFunction()/menu.c
    calls    :
           - send()/download.c
             Error()/download.c
             CheckError()/download.c
             InputBuffer()/download.c

*/
wmemory(step,width)
int step,width ;
{

char c,mm_code=0x05 ;
int i,j;

send(mm_code);
for(i=0;i<=400;i++);                    /* for Timing adjustment  */
if (!verify) send(c=(char)(width>>0)); /* send size of the operand*/
if (verify ) send(c=(char)((width|0x0080)>>0));
for(i=24;i>=0;i-=8) send(c=(char)(at>>i));
switch (width) {
case 4 : for(i=24;i>=0;i-=8) send(c=(char)(value>>i));
         if(verify) {
           InputBuffer(4);
           if((instring[0]!=(c=(char)(value>>24)))||
             (instring[1]!=(c=(char)(value>>16))) ||
             (instring[2]!=(c=(char)(value>>8 ))) ||
             (instring[3]!=(c=(char)(value>>0 ))) ) {
             Error("\pVerify Failed,Try Again","\p","\p","\p");
             at-=step*width;
             }
         } break;
case 2 : for(i=8;i>=0;i-=8)  send(c=(char)(value>>i));
```

```
        if(verify) {
          InputBuffer(2);
          if((instring[0]!=(c=(char)(value>>8))) ||
            (instring[1]!=(c=(char)(value>>0))) ) {
            Error("\pVerify Failed,Try Again","\p","\p","\p");
            at-=step*width;
            }
        } break;
case 1 : send(c=(char)(value>>0));
        if(verify){
          InputBuffer(1);
        if(instring[0]!=(c=(char)(value>>0))) {
          Error("\pVerify Failed,Try Again","\p","\p","\p");
          at-=step*width;
          } /* If Verify fails, do not increment/decrement the address */
        }break;
default : break;
}
at+=step*width;
CheckError();
ErrorFlag=0x00;
}


/*  GO()

    function:
           - This function performs the 'Go' operation. Program Counter,
             Trace Mode etc., are set by the user in Go Menu.
    arguments:

    called by:
           - doFunction()/menu.c
    calls    :
           - send()/download.c
             Stop_n_Flush()/download.c
             sendregs()/monitor.c
             SendFloat()/monitor.c
             CheckError()/download.c
             InputBuffer()/download.c
             CopyRegs()/download.c
             CopyBrkCnts()/download.c
             ltoa()/monitor.c
             Error()/download.c
             CopyFloat()/monitor.c

*/
go(){
char c,s[21],go_code=0x02;
```

```
int i,j;
long Save_PC,long_loc1=0,long_loc2=0;

Stop_n_Flush();
if(!WillGoTo) {
  go_code=0x03;
  Save_PC=registers[18];                    /* In case of Call */
  }
send(go_code);
for(i=0;i<=400;i++);
send(DisplaySteps);
                                            /* Send BreakPoints */
for(i=0;i<5;i++) {
   if(BreakTimes[i]==0) for(j=0;j<4;j++) send(0x00);
   if(BreakTimes[i]==1) for(j=24;j>=0;j-=8) send(c=(char)(Breaks[i]>>j));
   if(BreakTimes[i]>1)  for(j=24;j>=0;j-=8) send(c=(char)(Breaks[i]>>j));
}
                                            /* Send BreakCounts */
for(i=0;i<5;i++) {
    if(BreakTimes[i]>1)
      for(j=8;j>=0;j-=8) send(c=(char)(BreakTimes[i]>>j));
    else {
      send(0x00);
      send(0x00);
      }
    }
                                            /* Send Register info.  */
sendregs();                                 /* Send 68020 Registers */
if(Coprocessor) SendFloat();                /* Send 68881 Registers */
CheckError();
if(!ErrorFlag) {
  if(Coprocessor) InputBuffer(107+12+96);
  else
    InputBuffer(107);
    CopyRegs();                             /* Copy 68020 Registers */
    CopyBrkCnts();
    if(instring[106]==0x55) {
      ltoa(registers[18],s,8);
      Error("\pPrivilege violation  ","\p At address ",s,"\p ");
      }
     if(!WillGoTo) registers[18]=Save_PC;
     if(Coprocessor) CopyFloat(107);        /* Copy 68881 Registers */
     }
ErrorFlag=0x00;
Stop_n_Flush();
}


/*  HELP()
```

```
    function:
           - This function displays help information on the screen.
    arguments:

    called by:
           - doFunction()/menu.c
    calls    :
           - print()/monitor.c

*/
help(){
print("\p1- If you want to use Coprocessor instructions, you\n");
print("\p   need to select Coprocessor option. This can be done\n");
print("\p   in Options Menu.\n");
print("\p2- If you want to have a printout of what you see,\n");
print("\p   you need to select Hardcopy option. This can be done\n")
print("\p   in Options Menu.\n");
print("\p3- If you can not select Supervisor State to work in,\n");
print("\p   you need to select Experienced option. This can be done\n");
print("\p   in Options Menu.\n");
print("\p4- User is not allowed to set the Interrupt Level, to\n");
print("\p   a value greater than 3.\n");
print("\p5- If you suspect that your program, running on the\n");
print("\p   ECB, seems to be in an endless loop, or out of control,\n");
print("\p   press Abort Button on the ECB. In this case, you will\n");
print("\p   see the current register contents.\n");
print("\p6- If the solution in statement 5 above, won't work,\n");
print("\p   press Reset Button on the ECB. Also Reset Macintosh.\n");
}


/*  LTOA()

    function:
           - This function converts from long integer to Ascii.
    arguments:
           - l,s,len
    called by:
           - doFunction()/menu.c
           - CheckHex()/menu.c
           - CheckDec()/menu.c
           - go()/Monitor.c
           - printhex() /Monitor.c
           - printhex2()/Monitor.c
           - printhex3()/Monitor.c
           - CopyFloat()/Monitor.c
    calls    :
           - None

*/
```

```
ltoa(l,s,len)
char s[21];
long l;
int len;
{
int i;

for(i=s[0]=len;i>0;i--){
   s[i]=(l&0x0f)+'0';if(s[i]>'9') s[i]+=7;
   l=l>>4;
   }
}


/*  ITOA()

    function:
           - This function converts from  integer to Ascii.
    arguments:
           - n,s
    called by:
           - doFunction()/menu.c
    calls    :
           - None

*/
itoa(n,s)
char s[];
int n;
{
int i=1,c,k,l;
s[0]=4;
for(i=4;i>=1;i--) {
   if((n%10)==0) s[i]='0';
   else
     s[i]=n % 10 + '0';
   n/=10;
   }
}

/*  ATOI()

    function:
           - This function converts from  Ascii to integer.
    arguments:
           - s
    called by:
           - doFunction()/menu.c
    calls    :
           - None
```

```
*/
 int atoi(s)
 char s[];
{
iint i,n;
n=0;
for(i=1;i<=s[0];i++) n= 10 * n + s[i] - '0';
return(n);
}

/*  ATOL()

    function:
           - This function converts from  Ascii to long integer.
    arguments:
           - s
    called by:
           - doFunction()/menu.c
    calls    :
           - None

*/
long atol(s)
char s[21];
{
int i;
long l;

l=0;
for(i=1;i<=s[0];i++) {
   if(s[i]>'9') s[i]-=7;
   l=((s[i]-'0')&0x0f)+(l<<4);
   }
return(l);
}


/*  DOWNLOAD()

    function:
           - This function performs the 'Download' operation. First the
             user program is downloaded to Educational Computer Board.
             Then, the current register values, Coprocessor register values
             (if Coprocessor option is used), are received from the ECB.
    arguments:

    called by:
           - doFunction()/menu.c
    calls    :
```

```
            - send()/download.c
              Stop_n_Flush()/download.c
              CheckError()/download.c
              CopyRegs()/download.c
              CopyFloat()/monitor.c
              InputBuffer()/download.c

*/
DownLoad()
{
char *p,bite,down_code=0x00;
int chksum,i;
long l;

Stop_n_Flush();
if(Coprocessor)  down_code=0x08;        /* If Coprocessor is to be used */
send(down_code);
for(i=0;i<=400;i++);                    /* for Timing purposes */
for(p=start;p<end;p++) {
   if(p==start+8)  chksum=(bite=*p & 0xff) & 0xff ;
   if(p>start+8 )  chksum^=(bite=*p &0xff) & 0xff ;
   SerGetBuf(-6,&l);
   if(l>0) {
     ErrorFlag=1;
     break;
     }
   send(*p);
   }
if(!ErrorFlag) send(chksum);
CheckError();
if(!ErrorFlag) {
  if(Coprocessor) {
    InputBuffer(96+12+96);
    CopyRegs(); CopyFloat(96);
    }
    else {
      InputBuffer(96);
      CopyRegs();
      }
}
ErrorFlag=0x00;
Stop_n_Flush();
}

/* PRINTHEX()

    function:
           - This function prints onto the screen in hexadecimal format.
    arguments:
           - l,i
```

```
    called by:
           - DisAsm()/monitor.c
    calls    :
           - print()/monitor.c
             ltoa()/monitor.c

*/
printhex(l,i)
int i;
long l;
{
char s[21];
ltoa(l,s,i);
print(s); DrawChar(' ');
}


/* PRINT()

    function:
           - This function prints a string onto the screen.
    arguments:
           - s
    called by:
           - printhex()/monitor.c
           - help()/monitor.c
           - DumptoScreen()/monitor.c
           - DisAsm()/monitor.c
    calls    :
           - None

*/
print(s)
char s[];
{
int i;
Point p;
for(i=1;i<=s[0];i++) {
   if(s[i]=='\n') {
     ScrollRect(&myRect,0,-(scrollsize+4),myRgn);
     MoveTo(4,myRect.bottom-40);
     }
else
  DrawChar(s[i]);
}
}



/* PRINTHEX2()
```

```
    function:
           - This function, together with the  'Print2' function, copies th
             hexadecimal data to the 'PrintBuf'.
    arguments:
           - l,i,y
    called by:
           - DisAsm()/monitor.c
           - doFunction()/menu.c
    calls    :
           - print2()/monitor.c
             ltoa()/monitor.c

*/
prnthex2(l,i,y)
int i,y;
long l;
{
char s[21];
ltoa(l,s,i);
print2(s,y);
}


/* PRINT2()

    function:
           - This function, together with the  'printhex2' function,
             copies the hexadecimal data to the 'PrintBuf'.
    arguments:
           - s,y
    called by:
           - printhex2()/monitor.c
    calls    :
           - None

*/
print2(s,y)
char s[];
int y;
{
int i;
for(i=1;i<=s[0];i++,y++) {
   if(s[i]=='\n') break;
   else
     PrintBuf[y]=s[i];
}
}
```

```
/* PRINTHEX3()

    function:
           - This function, together with the  'Print3' function, copies the
             hexadecimal data to the 'PrintBuf'.
    arguments:
           - l,i,y
    called by:
           - dump()/monitor.c
           - memdisp()/menu.c
    calls    :
           - print3()/monitor.c
             ltoa()/monitor.c

*/
prnthex3(l,i,y)
int i,y;
long l;
{
char s[21];
ltoa(l,s,i);
print3(s,y);
}

/* PRINT3()

    function:
           - This function, together with the  'printhex3' function,
             copies the hexadecimal data to the 'PrintBuf'.
    arguments:
           - s,y
    called by:
           - printhex3()/monitor.c
    calls    :
           - None

*/
print3(s,y)
char s[];
int y;
{
int i;
for(i=1;i<=s[0];i++,y++){
   if(s[i]=='\n') break;
   else
     PrintBuf[y]=s[i];
   }
PrintBuf[y]=' '; y++;
z=y;
```

```
}


/*  SEND_REGS()

    function:
           - This function downloads all the MC68020 Data/Address/Control
             Register contents to the ECB.
    arguments:

    called by:
           - go()/Monitor.c
    calls    :
           - send()/download.c
             CheckError()/download.c

*/
sendregs()
{
char outchar;
int  m,chksum;
long tempbuf=0;
for(m=0;m<24;m++) {
   tempbuf=registers[m]&0xff000000;
   outchar=(char)(tempbuf>>24);
   if(m==0) chksum =(outchar & 0xff) & 0xff;
   else
   chksum ^=(outchar & 0xff) & 0xff;
   send(outchar);
   tempbuf=registers[m]&0x00ff0000;
   outchar=(char)(tempbuf>>16);
   chksum ^=(outchar & 0xff) & 0xff;
   send(outchar);
   tempbuf=registers[m]&0x0000ff00;
   outchar=(char)(tempbuf>>8);
   chksum ^=(outchar & 0xff) & 0xff;
   send(outchar);
   tempbuf=registers[m]&0x000000ff;
   outchar=(char)(tempbuf>>0);
   chksum ^=(outchar & 0xff) & 0xff;
   send(outchar);
   }
send((char)chksum);
CheckError();
ErrorFlag=0x00;
}

/*  DUMP_TO_PRN()
```

```
    function:
           - This function sends the contents of 'PrintBuf' to the printer.
    arguments:
           - index
    called by:
           - doFunction()/menu.c
           - dump()/Monitor.c
           - DisAsm()/Monitor.c
           - LastScreen()/Monitor.c
    calls    :
           - sendprn()/download.c

*/
DumptoPrn(index)
int index;
{
int i;
for(i=0;i<=index;i++)
   sendprn((char)(PrintBuf[i]));
}


/*  DUMP_TO_SCREEN()

    function:
           - This function sends the data, pointed to by 'ptr', to
             the screen.
    arguments:
           - index, ptr
    called by:
           - doFunction()/menu.c
           - dump()/Monitor.c
           - LastScreen()/Monitor.c
    calls    :
           - None

*/
DumptoScreen(index,ptr)
char *ptr;
int index;
{
char DrwStr[255];
int i,j=1;

for(i=0;i<=index;i++) {
   if(*ptr==0x0d) {
     DrwStr[0]=j-1;
     DrawString(DrwStr);j=1;
     print("\p\n");
```

```
      ptr +=2;  i++;
      }
    else {
      DrwStr[j]=(*ptr);
      ptr++; j++;
      }
    }
}



/*  DRAW_X()

    function:
           - This function writes the character into 'PrintBuf'.
    arguments:
           - y
    called by:
           - dump()/Monitor.c
    calls    :
           - None

*/
Draw_x(y)
int y;
{
PrintBuf[y]='x';
PrintBuf[y+1]='x';
PrintBuf[y+2]=' ';
z=y+3;
}


/*  DIS_ASM()

    function:
           - This function disassembles the code, which is passed to it.
    arguments:

    called by:
           - doFunction()/menu.c
    calls    :
           - send()/download.c
             print()/monitor.c
             printhex()/monitor.c
             FillQue()/monitor.c
             printhex2()/monitor.c
             DumptoPrn()/monitor.c
             InputBuffer()/download.c
```

```
*/
DisAsm()
{
char c,Dis_code=0x04 ;
int  Fixcount=12,i ;
asm {
     LEA    @44,A0 ;
     MOVE.L A0,HISPC;
     }
DisAsmOutBuf[0]=80;
*HISPC=StaDisAdr;
do {
   send(Dis_code);
   for(i=0;i<=400;i++);
   for(i=24;i>=0;i-=8) send(c=(char)(StaDisAdr>>i));
   send(c=(char)((Fixcount)>>0));
   InputBuffer(13);
   for(i=0;i<12;i++)  DisAsmInBuf[i]=instring[i];
   asm {
     MOVEM.L D0-D7/A0-A7,-(SP);
     BRA             @45;
@44: DC.L     0X00000000;
@45: LEA  DisAsmInBuf,A1;
     MOVE.L      (A1)+,D0;
     MOVE.L      (A1)+,D1;
     MOVE.L      (A1)+,D2;
     LEA DisAsmOutBuf,A1;
     ADD.L         #1,A1;
     MOVE.L       @44,A2;
     MOVE.L     A1,-(SP);
     MOVE.L     D0,-(SP);
     MOVE.L     D1,-(SP);
     MOVE.L     D2,-(SP);
     MOVE.L     A2,-(SP);
     MOVE.L   SubrAdr,A3;
     JSR             (A3);
     MOVE.L     (SP)+,A2;
     LEA          @44,A3;
     MOVE.L      A2,(A3);
     ADD.L        #16,SP;
     MOVEM.L (SP)+,D0-D7/A0-A7;
   }
   printhex(StaDisAdr,8);
   print(DisAsmOutBuf);
   print("\p\n");
   if(NotAfterGo) {
     i=0;
     prnthex2(StaDisAdr,8,i);
     i+=8;
```

```
      PrintBuf[8]=' ';
      for(i=9;i<88;i++) PrintBuf[i]=DisAsmOutBuf[i-8];
      PrintBuf[i]=0x0d;i++;
      PrintBuf[i]=0x0a;
      FillQue(i+1);
      if(Hardcopy) DumptoPrn(i);
      }
    StaDisAdr= *HISPC;
    } while (StaDisAdr <= EndDisAdr);
NotAfterGo=0;
}

/*  COPY_FLOAT()

    function:
          - This function copies the Floating Point Registers, which
            are uploaded by the ECB.
    arguments:
          - fmWhere
    called by:
          - DownLoad()/Monitor.c
          - go()/Monitor.c
    calls    :
          - ltoa()/monitor.c
          - Error()/download.c

*/
CopyFloat(fmWhere)
int fmWhere;
{
char instring2[4],NotNumber=0,s[21];
int i,j,k,p=0,r;
r=fmWhere;
while(p<3) {       /* First Copy Coprocessor's control registers */
     fcregs[p]= 0;
     for (j=0;j<4;j++) {
          instring2[j]=instring[r];
          r++;
          }
     for(j=0;j<4;j++) fcregs[p]=(instring2[j]&0xff)+(fcregs[p]<<8);
     p++;
     }
fmWhere=r;
for(j=0;j<8;j++) {
   for(i=0;i<12;i++) Fbuf[i]=instring[fmWhere+i+j*12];
   fregs[0][j]=Fbuf[3]+0x30;
   fregs[17][j]=(Fbuf[0]&0x0f)+0x30;
   fregs[18][j]=((Fbuf[1]>>4)&0x0f)+0x30;
   fregs[19][j]=(Fbuf[1]&0x0f)+0x30;
   for(r=17;r<20;r++)
```

```
      if((fregs[r][j]<0x30)||(fregs[r][j]>0x39)) {
        for (p=0;p<20;p++) fregs[p][j]=0x30;
        NotNumber=1;
        ltoa((long)(j),s,2);
        Error("\pNot A Number                ","\por Infinity
        ",
           "\pIn FPReg. # ",s);
        }
if(!NotNumber) {
  if((Fbuf[0]&0x80)!=0) ManSign[j]='-';
else
  ManSign[j]='+';
if((Fbuf[0]&0x40)!=0) ExpSign[j]='-';
else
  ExpSign[j]='+';
for(i=1,k=4; k<12; i+=2,k++) {
   fregs[i][j]  =((Fbuf[k]>>4)&0x0f)+0x30;
   fregs[i+1][j]=(Fbuf[k]&0x0f)+0x30;
   }
  }
NotNumber=0;
}
}


/*  SEND_FLOAT()

    function:
           - This function downloads the Floating Point Registers, to the ECB
    arguments:

    called by:
           - go()/Monitor.c
    calls    :
           - send()/download.c

*/
SendFloat()

{
char outchar,chksum;
int i,j,k;
long tempbuf;

for(j=0;j<3;j++) {           /* First Send Control,Staus,I Registers */
   tempbuf=fcregs[j]&0xff000000;
   outchar=(char)(tempbuf>>24);
   if(j==0) chksum =(outchar & 0xff) & 0xff;
   else
     chksum ^=(outchar & 0xff) & 0xff;
```

```
    send(outchar);
    tempbuf=fcregs[j]&0x00ff0000;
    outchar=(char)(tempbuf>>16);
    chksum ^=(outchar & 0xff) & 0xff;
    send(outchar);
    tempbuf=fcregs[j]&0x0000ff00;
    outchar=(char)(tempbuf>>8);
    chksum ^=(outchar & 0xff) & 0xff;
    send(outchar);
    tempbuf=fcregs[j]&0x000000ff;
    outchar=(char)(tempbuf>>0);
    chksum ^=(outchar & 0xff) & 0xff;
    send(outchar);                          /* Control registers are sent */
    }
for(j=0;j<8;j++) {
    if(ManSign[j]=='+') Fbuf[0]=0x00;
    else
      Fbuf[0]=0x80;
    if(ExpSign[j]=='+') Fbuf[0]=Fbuf[0] & 0xbf;
    else
      Fbuf[0]=Fbuf[0] | 0x40;
    Fbuf[0]=Fbuf[0]|(fregs[17][j] - 0x30);
    Fbuf[1]=( ((fregs[18][j]-0x30)<<4)|(fregs[19][j]-0x30) );
    Fbuf[2]=0x00;
    Fbuf[3]=fregs[0][j]-0x30;
    for(i=4,k=1;i<12;i++,k+=2)
       Fbuf[i]=((fregs[k][j]-0x30)<<4)|(fregs[k+1][j]-0x30);
    for(k=0;k<12;k++) {
       send(Fbuf[k]);                   /* Send Floating Point Registers */
       chksum ^=(Fbuf[k] & 0xff) & 0xff;
       }
}
send(chksum);
}


/*  LAST_SCREEN()

    function:
           - This function displays the latest screen-full information.
    arguments:
           - k
    called by:
           - doFunction()/menu.c
           - HandleEvent()/download.c
    calls    :
           - DumptoPrn()/monitor.c
           - DumptoScreen()/monitor.c

*/
```

```
LastScreen(k)
int k;
{
int i,lineNum;
long difference;
if(k!=2) {
  Tail=Head;
  lineNum=0;
  for(;;) {
     if(lineNum>20) {
       Tail+=2; break;
       }
     if(Tail< StartQue) {
       Tail=EndQue;
       if(Reach) {
         Tail=StartQue;break;
         }
     }
   if((*Tail)==0x0a) lineNum++;
  Tail--;
}
difference=Head-Tail+1L;
if(difference<0)
  difference=((EndQue-Tail)+(Head-StartQue))+2L;
for(i=0;i<(int)(difference);i++) {
   PrintBuf[i]=*Tail;
   Tail++;
   if(Tail>EndQue) Tail=StartQue;
   }
EraseRect(&myRect);
DumptoScreen(((int)(difference)-2),&PrintBuf[0]);
if(k) DumptoPrn(((int)(difference)-2),&PrintBuf[0]);
 }
}


/*  FILL_QUE()

    function:
           - This function adds the latest data to the circular queue.
    arguments:
           - index
    called by:
           - doFunction()/menu.c
           - dump()/Monitor.c
           - DisAsm()/Monitor.c
    calls    :
           - None

*/
```

```
FillQue(index)
int index;
{
int i;

for(i=0;i<index;i++) {
   if(Head>EndQue) {
     Head=StartQue;
     Reach=0;
     }
   *Head=PrintBuf[i];
   Head++;
   }
}
```

iv. Source code of disasm.c

```
Dassy()

{
extern long *SubrAdr;

/*
BUFSIZE   EQU       80                ;SIZE OF OUTPUT BUFFER
EOT       EQU       4                 ;
FDATA     EQU       4                 ;DATA FIELD
FOC       EQU       31                ;OP-CODE FIELD
FOP       EQU       39                ;OPERAND FIELD
LOCVARSZ  EQU       16
                     */
asm{
          MOVEM.L D0-D7/A0-A7,-(SP);
          LEA  @DECODE,A0  ;
          MOVE.L A0,SubrAdr;
          LEA  @IS2,A0     ;THE FOLLOWING CODE (UNTIL THE LINE
          LEA  @PGM,A4     ; /* DISASSEMBLY PROGRAM BEGINS */),
          LEA  @ISHIFT,A5  ; CALCULATES THE DISPLACEMENT OF A
          LEA  @ISH1,A6    ; ROUTINE HANDLING ANY PARTICULAR
          BSR  @SUBR       ; INSTRUCTION (SUCH AS MOVE, ADD ETC.)
          LEA  @ISH2,A6    ; FROM THE BEGINNING OF THE PROGRAM.
          BSR  @SUBR       ; THIS DISPLACEMENT VALUE IS THEN
          LEA  @ISH3,A6    ; WRITTEN INTO THE CORRESPONDING
          BSR  @SUBR       ; ENTRY IN TABLE "@TBL".
          LEA  @ISH4,A6    ;
          BSR  @SUBR       ;
          LEA  @ISH5,A6    ;
          BSR  @SUBR       ;
          LEA  @ISH6,A6    ;
          BSR  @SUBR       ;
          LEA  @ISH7,A6    ;
          BSR  @SUBR       ;
          LEA  @ISH8,A6    ;
          BSR  @SUBR       ;

          LEA  @FORM10EX,A5;
          LEA  @F10EX1,A6  ;
          BSR  @SUBR       ;
          LEA  @F10EX2,A6  ;
          BSR  @SUBR       ;
          LEA  @F10EX3,A6  ;
          BSR  @SUBR       ;
          LEA  @F10EX4,A6  ;
          BSR  @SUBR       ;
```

```
LEA   @F10EX5,A6   ;
BSR   @SUBR        ;
LEA   @F10EX6,A6   ;
BSR   @SUBR        ;

LEA   @FORM12,A5   ;
LEA   @F121,A6     ;
BSR   @SUBR        ;
LEA   @F122,A6     ;
BSR   @SUBR        ;
LEA   @F123,A6     ;
BSR   @SUBR        ;
LEA   @F124,A6     ;
BSR   @SUBR        ;

LEA   @FORM9,A5    ;
LEA   @F91,A6      ;
BSR   @SUBR        ;

LEA   @FORM8,A5    ;
LEA   @F81,A6      ;
BSR   @SUBR        ;

LEA   @FORM7,A5    ;
LEA   @F71,A6      ;
BSR   @SUBR        ;

LEA   @FORM6D,A5   ;
LEA   @F6D1,A6     ;
BSR   @SUBR        ;
LEA   @F6D2,A6     ;
BSR   @SUBR        ;
LEA   @F6D3,A6     ;
BSR   @SUBR        ;
LEA   @F6D4,A6     ;
BSR   @SUBR        ;
LEA   @F6D5,A6     ;
BSR   @SUBR        ;

LEA   @FORM10,A5   ;
LEA   @F101,A6     ;
BSR   @SUBR        ;
LEA   @F102,A6     ;
BSR   @SUBR        ;
LEA   @F103,A6     ;
BSR   @SUBR        ;

LEA   @FORM12A,A5  ;
LEA   @F12A1,A6    ;
BSR   @SUBR        ;
```

```
LEA   @IMOVEQ,A5   ;
LEA   @IMVQ1,A6    ;
BSR   @SUBR        ;

LEA   @IBSR,A5     ;
LEA   @IBSR1,A6    ;
BSR   @SUBR        ;
LEA   @IBSR2,A6    ;
BSR   @SUBR        ;

LEA   @ICC,A5      ;
LEA   @ICC1,A6     ;
BSR   @SUBR        ;

LEA   @IDBCC,A5    ;
LEA   @IDBCC1,A6   ;
BSR   @SUBR        ;

LEA   @SCC,A5      ;
LEA   @SCC1,A6     ;
BSR   @SUBR        ;

LEA   @IQUICK,A5   ;
LEA   @IQUICK1,A6  ;
BSR   @SUBR        ;
LEA   @IQUICK2,A6  ;
BSR   @SUBR        ;

LEA   @FORM6A,A5   ;
LEA   @F6A1,A6     ;
BSR   @SUBR        ;

LEA   @FORM11SL,A5;
LEA   @F11SL1,A6   ;
BSR   @SUBR        ;
LEA   @F11SL2,A6   ;
BSR   @SUBR        ;

LEA   @SCOMMON,A5  ;
LEA   @SCOMMON1,A6;
BSR   @SUBR        ;
LEA   @SCOMMON2,A6;
BSR   @SUBR        ;
LEA   @SCOMMON3,A6;
BSR   @SUBR        ;
LEA   @SCOMMON4,A6;
BSR   @SUBR        ;
LEA   @SCOMMON5,A6;
BSR   @SUBR        ;
```

```
LEA   @SCOMMON6,A6;
BSR   @SUBR       ;

LEA   @ISTOP,A5   ;
LEA   @ISTOP1,A6  ;
BSR   @SUBR       ;

LEA   @IMVFUSP,A5 ;
LEA   @IMVFUSP1,A6;
BSR   @SUBR       ;

LEA   @IMVTUSP,A5 ;
LEA   @IMVTUSP1,A6;
BSR   @SUBR       ;

LEA   @FORM5,A5   ;
LEA   @F51,A6     ;
BSR   @SUBR       ;

LEA   @FORM4,A5   ;
LEA   @F41,A6     ;
BSR   @SUBR       ;

LEA   @ILINK,A5   ;
LEA   @ILINK1,A6  ;
BSR   @SUBR       ;

LEA   @IMOVEMTR,A5;
LEA   @IMVMTR1,A6 ;
BSR   @SUBR       ;

LEA   @FORM1A,A5  ;
LEA   @F1A1,A6    ;
BSR   @SUBR       ;
LEA   @F1A2,A6    ;
BSR   @SUBR       ;

LEA   @FORM1,A5   ;
LEA   @F11,A6     ;
BSR   @SUBR       ;
LEA   @F12,A6     ;
BSR   @SUBR       ;
LEA   @F13,A6     ;
BSR   @SUBR       ;
LEA   @F14,A6     ;
BSR   @SUBR       ;
LEA   @F15,A6     ;
BSR   @SUBR       ;

LEA   @FORM3,A5   ;
```

```
LEA   @F31,A6      ;
BSR   @SUBR        ;
LEA   @F32,A6      ;
BSR   @SUBR        ;
LEA   @F33,A6      ;
BSR   @SUBR        ;

LEA   @IMOVEMFR,A5;
LEA   @IMVMFR1,A6 ;
BSR   @SUBR        ;

LEA   @FORM11,A5   ;
LEA   @F111,A6     ;
BSR   @SUBR        ;

LEA   @IMVTSR,A5   ;
LEA   @IMVTSR1,A6  ;
BSR   @SUBR        ;

LEA   @IMVTCCR,A5  ;
LEA   @IMVTCCR1,A6;
BSR   @SUBR        ;

LEA   @IMVFSR,A5   ;
LEA   @IMVFSR1,A6  ;
BSR   @SUBR        ;

LEA   @IMOVE,A5    ;
LEA   @IMOVE1,A6   ;
BSR   @SUBR        ;
LEA   @IMOVE2,A6   ;
BSR   @SUBR        ;
LEA   @IMOVE3,A6   ;
BSR   @SUBR        ;

LEA   @IMMED,A5    ;
LEA   @IMMED1,A6   ;
BSR   @SUBR        ;
LEA   @IMMED2,A6   ;
BSR   @SUBR        ;
LEA   @IMMED3,A6   ;
BSR   @SUBR        ;
LEA   @IMMED4,A6   ;
BSR   @SUBR        ;
LEA   @IMMED5,A6   ;
BSR   @SUBR        ;
LEA   @IMMED6,A6   ;
BSR   @SUBR        ;

LEA   @IMOVEP,A5   ;
```

```
            LEA   @IMOVEP1,A6 ;
            BSR   @SUBR       ;

            LEA   @ISETS,A5   ;
            LEA   @ISETS1,A6  ;
            BSR   @SUBR       ;
            LEA   @ISETS2,A6  ;
            BSR   @SUBR       ;
            LEA   @ISETS3,A6  ;
            BSR   @SUBR       ;
            LEA   @ISETS4,A6  ;
            BSR   @SUBR       ;

            LEA   @ISETD,A5   ;
            LEA   @ISETD1,A6  ;
            BSR   @SUBR       ;
            LEA   @ISETD2,A6  ;
            BSR   @SUBR       ;
            LEA   @ISETD3,A6  ;
            BSR   @SUBR       ;
            LEA   @ISETD4,A6  ;
            BSR   @SUBR       ;
            JMP  (A0)         ;
@SUBR:      MOVE.L A5,A3      ;
            SUB.L  A4,A3      ;
            MOVE.W A3,(A6)    ;
            RTS               ;


/*  DISASSEMBLY PROGRAM BEGINS    */
/*  CALLING SEQUENCE:
    D0,D1,D2 Contains the code to be Disassembled
    A4 = Value of Program Counter for the code
    A5 = Pointer to store data (BUFSIZE = 80 assumed)
    JSR   DECODE
    RETURN:
    A4 = Value of Program Counter for next instruction
    A5 = Pointer to line as Disassembled
    A6 = Pointer to End Of Line

    01234567890123456789012345678901234567890123456789
    AAAAAA FDATA.DDDDDDDDDDDDDD FOC.... FOP.....
*/
@PGM:       NOP                           ;BASE ADDRESS THIS MODULE

/*  MOVEM REGISTERS TO EA

            01001D001S......
            ..........XXXXXX              EFFECTIVE ADDRESS
```

```
          .........0.......           WORD
          .........1.......           LONG
          .....0...........           REGISTER TO MEMORY
          .....1...........           MEMORY TO REGISTER
*/
@IMOVEMFR:BSR       @MOVEMS             ;SIZE
          MOVE.L    #0X0038,D6          ;.                                  1,4
          AND.W     (A4),D6             ;.                                  1,4
          CMP.W     #0X0020,D6          ;
          BEQ.S     @IM7788             ;PREDECREMENT MODE
          MOVE.L    #1,D6               ;D6 = INCREMENTER (BIT POSITION)
          MOVE.L    #0,D1               ;D1 = BIT POSITION
          BRA.S     @IM7799             ;
@IM7788:  MOVE.L    #-1,D6              ;D6 = DECREMENTER (BIT POSITION)
          MOVE.L    #15,D1              ;D1 = BIT POSITION
@IM7799:  BSR       @MOVEMR             ;BUILD MASK WORD
          MOVE.B    #',',(A6)+          ;STORE COMMA
          ADD.L     #2,D3               ;
          MOVE.W    (A4),D4             ;
          MOVE.W    #0X1F4,D7           ;CONTROL + PREDECREMENT
          BSR       @EA                 ;
          BRA.S     @CS16               ;COMMON

*         MOVEM  EA  TO REGISTERS       */
@IMOVEMTR:BSR       @MOVEMS             ;SIZE
          ADD.L     #2,D3               ;
          MOVE.W    #0X7EC,D7           ;CONTROL + POSTINCREMENT
          BSR       @EA                 ;
          MOVE.B    #',',(A6)+          ;STORE COMMA
          MOVE.L    #1,D6               ;D6 = BIT POSITION INCREMENTER
          MOVE.L    #0,D1               ;D1 = BIT POSITION
          BSR       @MOVEMR             ;
CS16:     BRA       @CS15               ;COMMON
ISTOP:    MOVE.W    2(A4),D0            ;
          MOVE.B    #'#',(A6)+          ;IMMEDIATE
          MOVE.B    #'$',(A6)+          ;HEX
          BSR       @PNT4HX             ;VALUE
          BRA       @COMMON4            ;
IMMED:    BSR       @FORMSIZE           ;ADD  AND  CMP #  EOR  OR  SUB
          ADD.L     #2,D3               ;SIZE = 4
          MOVE.B    #'#',(A6)+          ;IMMEDIATE
          CLR.L     D0                  ;
          MOVE.W    2(A4),D0            ;D0 = EXTENSION WORD
          MOVE.W    (A4),D1             ;
          LSR.W     #6,D1               ;
          AND.W     #3,D1               ;
          BEQ.S     @IMMED65            ;BYTE
          CMP.B     #1,D1               ;
          BEQ.S     @IMMED75            ;WORD
          ADD.L     #2,D3               ;.LONG    SIZE = 6
```

```
            MOVE.L    2(A4),D0          ;D0 = LONG EXTENSION WORD
@IMMED45:   BSR       @HEX2DEC          ;DECIMAL
            MOVE.B    D5,(A6)+          ;COMMA SEPARATOR
            MOVE      (A4),D0           ;
            AND.W     #0X003F,D0        ;
            CMP.W     #0X003C,D0        ;DESTINATION ADDRESS MODE 111100 "SR
            BNE.S     @IMMED55          ;NOT FOUND
            MOVE.W    (A4),D0           ;"SR"  ILLEGAL FOR
            AND.W     #0X4000,D0        ;ADDI   SUBI  CMPI
            BNE       @FERROR           ;0600   0400  0C00
            MOVE.W    (A4),D1           ;
            AND.W     #0X00C0,D1        ;
            CMP.W     #0X0080,D1        ;
            BEQ       @FERROR           ;.LONG NOT ALLOWED
            MOVE.W    (A4),D1           ;
            BTST.L    #6,D1             ;
            BNE       @STAT             ;
            MOVE.B    #'C',(A6)+        ;#, CCR FOR ANDI, EORI, ORI
            MOVE.B    #'C',(A6)+        ;
            MOVE.B    #'R',(A6)+        ;
            BRA.S     @CS14             ;COMMON
@STAT:      MOVE.B    #'S',(A6)+        ;#,SR FOR ANDI, EORI, ORI
            MOVE.B    #'R',(A6)+        ;
@CS15:      BRA.S     @CS14             ;COMMON
@IMMED55:   BSR       @EA               ;
            BRA.S     @CS14             ;COMMON
@IMMED65:   MOVE.L    D0,D1             ;D1 = XXXXXXXX........
            LSR.W     #8,D1             ;D1 = 00000000XXXXXXXX
            BEQ.S     @IMMED75          ;
            MOVE.L    D0,D1             ;
            ASR.W     #7,D1             ;
            ADD.W     #1,D1             ;CHECK FOR NEGATIVE
            BNE       @FERROR           ;
IMMED75:    EXT.L     D0                ;
            BRA       @IMMED45          ;

/*          BIT  5432109876543210
            ....RRRMMM....              DESTINATION REGISTER MODE
            ..........MMMRRR            SOURCE MODE REGISTER
            0001............            .BYTE
            0011............            .WORD
            0010............            .LONG
            IF BYTE SIZE, DESTINATION ADDRESS DIRECT NOT ALLOWED.
*/
@IMOVE:     BRA       @IMOVEA1          ;
@ILINK:     BSR.S     @FORMREGA         ;
            MOVE.B    D5,(A6)+          ;COMMA SEPARATOR
            MOVE.B    #'#',(A6)+        ;
            MOVE.W    2(A4),D0          ;
            EXT.L     D0                ;
```

```
          BSR       @HEX2DEC          ;DECIMAL DISPLACEMENT
          BRA       @COMMON4          ;
@FORM1:   BSR       @FORMSIZE         ;CLR  NEG  NEGX  NOT TST

/*        NBCD      TAS               */
@FORM1A:  BSR       @EA               ;DATA ALTERABLE ONLY
@CS14:    BRA       @CS13             ;COMMON
@FORM3:   BSR.S     @FORMREGD         ;EXT  SWAP
          BRA.S     @CS13             ;COMMON
@FORM4:   MOVE.B    #('#'),(A6)+      ;TRAP
          MOVE.W    (A4),D0           ;
          AND.L     #0X0F,D0          ;
          BSR       @HEX2DEC          ;DECIMAL
          BRA.S     @CS13             ;COMMON
@FORM5:   BSR.S     @FORMREGA         ;UNLNK
          BRA.S     @CS13             ;COMMON

/*        5432109876543210
          ....RRR.........          ADDRESS REGISTER
          ..........XXXXXX          EFFECTIVE ADDRESS
*/
@FORM6A:  MOVE.W    #0X7E4,D7         ;CONTROL ADDRESSING
          BSR.S     @EA10             ;
          MOVE.B    D5,(A6)+          ;COMMA SEPARATOR
          MOVE.W    (A4),D4           ;
          ROL.W     #7,D4             ;
          BSR.S     @FORMREGA         ;
          BRA.S     @CS13             ;COMMON

/*        BIT  5432109876543210
          ....DDD.........          DATA REGISTER
          ..........XXXXXX          EFFECTIVE ADDRESS
*/
@FORM6D:  MOVE.W    #0XFFD,D7         ;CHK DIVS DIVU MULS MULU DATA ,ADRESG
          BSR.S     @EA10             ;
          MOVE.B    D5,(A6)+          ;COMMA SEPARATOR
          MOVE.W    (A4),D4           ;
          ROL.W     #7,D4             ;
          BSR.S     @FORMREGD         ;
          BRA.S     @CS13             ;COMMON
@FORMREGA:MOVE.B    #'A',(A6)+        ;FORMAT A@
@FORMREG5:AND.B     #0X07,D4          ;
          OR.B      #('0'),D4         ;
          MOVE.B    D4,(A6)+          ;
          RTS                         ;
@FORMREGD:MOVE.B    #('D'),(A6)+      ;FORMAT D@
          BRA       @FORMREG5         ;
/*        BIT  5432109876543210
          ....DDD......DDD          DATA REGISTERS
*/
```

```
@FORM7:    ROL.W     #7,D4              ;EXG
           BSR       @FORMREGD          ;
           MOVE.B    D5,(A6)+           ;COMMA SEPARATOR
           MOVE.W    (A4),D4            ;
           BSR       @FORMREGD          ;
           BRA.S     @CS13              ;COMMON

/*         BIT  5432109876543210
           ....AAA......AAA             ADDRESS REGISTERS
*/
@FORM8:    ROL.W     #7,D4              ;EXG
           BSR       @FORMREGA          ;
@FORM815:  MOVE.B    #',',(A6)+         ;COMMA SEPARATOR
           MOVE.W    (A4),D4            ;
           BSR       @FORMREGA          ;
@CS13:     BRA       @CS12              ;COMMON

/*         BIT  5432109876543210
           ....DDD.........             DATA REGISTER
           ............AAA              ADDRESS REGISTER
*/
@FORM9:    ROL.W     #7,D4              ;EXG
           BSR       @FORMREGD          ;DATA REGISTER
           BRA       @FORM815           ;
@EA10:     BRA       @EA                ;

/*         5432109876543210
           ..........AAAAAA             EFFECTIVE ADDRESS
           .......MMM......             OP-MODE
           ....RRR.........             D-REGISTER
           .......011......             WORD  EA, A@
           .......111......             LONG  EA, A@
           .......000......              EA, D@ BYTE (ADDRESS REGISTER DIREC
                                        NOT ALLOWED)
           .......0........             EA, D@
           .......1........             D@, EA
           ........00......             BYTE
           ........01......             WORD
           ........10......             LONG

           ADD <EA>,A@   CMP <EA>,A@   SUB <EA>,A@
*/
@FORM10EX:MOVE.W     #0XFFF,D7          ;ADD  CMP  SUB,ALL MODES ALLOWED
           MOVE.L    D4,D0              ;
           AND.W     #0X01C0,D0         ;
           BEQ.S     @FORM103           ;.......000......
           CMP.W     #0X01C0,D0         ;
           BEQ.S     @FORM10E3          ;.......111......
           CMP.W     #0X00C0,D0         ;
           BNE.S     @FORM10E6          ;
```

```
          MOVE.B    #'.',(A5)+          ;.......011.....       STORE PERIOD
          MOVE.B    #'W',(A5)+          ;
          BRA.S     @FORM10E4           ;
@FORM10E3:MOVE.B    #'.',(A5)+          ;
          MOVE.B    #'L',(A5)+          ;
@FORM10E4:BSR       @EA10               ;
          MOVE.B    D5,(A6)+            ;STORE COMMA SEPARATOR
          MOVE.W    (A4),D4             ;
          ROL.W     #7,D4               ;
          BSR       @FORMREGA           ;<EA>,A@
          BRA.S     @CS12               ;COMMON
@FORM10E6:BTST.B    #0,(A4)             ;
          BNE.S     @FORM105            ;.......1........    D@,<EA>
          BRA.S     @FORM104            ;.......0........    <EA>,D@

/*        5432109876543210
          ..........AAAAAA              EFFECTIVE ADDRESS
          .......MMM......              OP-MODE
          ....RRR.........              D-REGISTER
          .......0........              EA,D@
          .......1........              D@,EA
          ........00......              BYTE
          ........01......              WORD
          ........10......              LONG
*/
@FORM10:  BTST.B    #0,(A4)             ;AND  EOR  OR
          BNE.S     @FORM105            ;
@FORM103: MOVE.W    #0XFFD,D7           ;DATA ADDRESSING
@FORM104: BSR       @FORMSIZE           ;
          BSR       @EA10               ;<EA>,D@
          MOVE.B    D5,(A6)+            ;COMMA SEPARATOR
          MOVE.B    (A4),D4             ;
          LSR.B     #1,D4               ;
          BSR       @FORMREGD           ;
          BRA.S     @CS12               ;COMMON
@FORM105: BSR       @FORMSIZE           ;D@,<EA>
          MOVE.B    (A4),D4             ;
          LSR.B     #1,D4               ;
          BSR       @FORMREGD           ;
          MOVE.B    D5,(A6)+            ;COMMA SEPARATOR
          MOVE.W    (A4),D4             ;
          MOVE.W    #0X1FD,D7           ;ALTERABLE MEMORY ADDRESSING

/*        PEA     (JMP  JSR)            */
          BSR       @EA10               ;
@CS12:    BRA       @COMMON             ;
@FORM11:  MOVE.W    #0X7E4,D7           ;CONTROL ADDERSSING
          BSR       @EA10               ;
          BRA.S       @CS12             ;COMMON
```

```
/*          JMP  JSR                          */
@FORM11SL:MOVE.L     D4,D0                    ;
          AND.W      #0X3F,D0                 ;
          CMP.W      #0X38,D0                 ;
          BNE.S      @FORM112                 ;
          MOVE.B     #'.',(A5)+               ;
          MOVE.B     #'S',(A5)+               ;
@FORM112: CMP.W      #0X39,D0                 ;
          BNE.S      @FORM114                 ;
          MOVE.B     #'.',(A5)+               ;
          MOVE.B     #'L',(A5)+               ;
@FORM114: BRA        @FORM11                  ;

/*        BIT 5432109876543210
          ....XXX.....0...                    DATA DESTINATION REGISTER
          ....XXX.....1...                    ADDRESS REGISTER
          ....XXX.00......                    BYTE
          ........01......                    WORD
          ........10......                    LONG
          ............0...                    DATA REGISTER TO DATA REGISTER
          ............1...                    MEMORY TO MEMORY
          ............0XXX                    DATA SOURCE REGISTER
          ............1XXX                    ADDRESS SOURCE REGISTER
*/
@FORM12:  BSR        @FORMSIZE                ;ABCD  ADDX  SBCD  SUBX
          BTST       #3,D4                    ;
          BNE.S      @FORM125                 ;
          BSR        @FORMREGD                ;D@,D@ FORMAT SOURCE
          MOVE.B     D5,(A6)+                 ;COMMA SEPARATOR
          MOVE.B     (A4),D4                  ;
          LSR.B      #1,D4                    ;
          BSR        @FORMREGD                ;FORMAT DESTINATION
          BRA.S      @CS11                    ;COMMON
@FORM125: MOVE.B     #'-',(A6)+               ;
          MOVE.B     #'(',(A6)+               ;
          BSR        @FORMREGA                ;
          MOVE.L     #0X282D2C29,D0           ;'(-,)'
          BSR.S      @SCHR                    ;
          MOVE.B     (A4),D4                  ;
          LSR.B      #1,D4                    ;
          BSR        @FORMREGA                ;
          MOVE.B     #')',(A6)+               ;
          BRA.S      @CS11                    ;

/*        BIT 5432109876543210
          ....XXX.....1...                    ADDRESS REGISTER    DESTINATION
          ....XXX.00......                    BYTE
          ........01......                    WORD
          ........10......                    MLONG
          ............1...                    MEMORY TO MEMORY
```

```
          ............1XXX              ADDRESS SOURCE REGISTER
*/
@FORM12A: BSR       @FORMSIZE           ;CMPM
          MOVE.B    #'(',(A6)+          ;(
          BSR       @FORMREGA           ;A@
          MOVE.L    #0X282C2B29,D0      ; '(,+)'
          BSR.S     @SCHR               ;STORE CHARS
          MOVE.B    (A4),D4             ;
          LSR.B     #1,D4               ;
          BSR       @FORMREGA           ;A@
          MOVE.B    #')',(A6)+          ;
          MOVE.B    #'+',(A6)+          ;
@CS11:    BRA       @COMMON             ;
@IQUICK:  BRA       @IQUICKA            ;ADDQ  SUBQ
/*        BIT 5432109876543210
          0111...0........              FIXED
          ....RRR.........              DATA REGISTER
          ........DDDDDDDD              SIGN EXTENDED DATA
*/
@IMOVEQ:  MOVE.B    #'#',(A6)+          ;IMMEDIATE
          MOVE.W    (A4),D0             ;
          EXT.W     D0                  ;
          EXT.L     D0                  ;
          BSR       @HEX2DEC            ;DECIMAL
          MOVE.B    D5,(A6)+            ;COMMA SEPARATOR
          ROL.W     #7,D4               ;
          BSR       @FORMREGD           ;
          BRA       @CS11               ;COMMON
@SCHR:    MOVE.B    D0,(A6)+            ;OUTPUT STRING
          LSR.L     #8,D0               ;
          BNE       @SCHR               ;MORE TO OUTPUT
          RTS                           ;

/*        MOVE FROM STATUS REGISTER (SR)  */
@IMVFSR:  MOVE.L    #(0X2C5253),D0      ;',RS'  SR,
          BSR       @SCHR               ;
          BSR       @EA                 ;DATA ALTERABLE
          BRA       @CS11               ;COMMON

/*        MOVE FROM USP (USER STACK POINTER)  */
@IMVFUSP: MOVE.L    #(0X2C505355),D0    ;USP, ",PSU"
          BSR       @SCHR               ;
          BSR       @FORMREGA           ;
          BRA       @CS11               ;COMMON

/*        MOVE TO SR (STATUS REGISTER) */
@IMVTSR:  MOVE.W    #0XFFD,D7           ;DATA ADDRESSING
          BSR       @EA                 ;
          MOVE.L    #(0X52532C),D0      ;SR "RS,"
@IMVT44:  BSR       @SCHR               ;
```

```
          BRA       @CS11               ;COMMON

/*        MOVE TO USP (USER STACK POINTER)  */
@IMVTUSP: BSR       @FORMREGA
          MOVE.L    #(0X5053552C),D0    ;,USP "PSU,"
          BRA       @IMVT44             ;

/*        MOVE TO CCR (CONDITION CODE REGISTER)  */
@IMVTCCR: MOVE.W    #0XFFD,D7           ;DATA ADDRESSING
          BSR       @EA                 ;
          MOVE.L    #(0X5243432C),D0    ;,CCR "RCC,"
          BRA       @IMVT44

/*        BIT  5432109876543210
          0000...1..001...              FIXED
          ....XXX.........              DATA REGISTER
          ........0.......              MEMORY TO REGISTER
          ........1.......              REGISTER TO MEMORY
          .........0......              WORD
          .........1......              LONG
          ............XXX               ADDRESS REGISTER
*/
@IMOVEP:  MOVE.B    #'.',(A5)+          ;D@,#(A@)
          MOVE.W    #(0X4C57),D0        ; "LW"
          BTST      #6,D4               ;
          BEQ.S     @IMOVEP11           ;USE "W"
          LSR.W     #8,D0               ;USE "L"
@IMOVEP11:MOVE.B    D0,(A5)+            ;LENGTH
          MOVE.B    (A4),D4             ;
          LSR.B     #1,D4               ;
          BTST.B    #7,1(A4)            ;
          BEQ.S     @IMOVEP35           ;
          BSR       @FORMREGD           ;D@,0XHHHH(A@)
          MOVE.B    D5,(A6)+            ;COMMA SEPARATOR
          MOVE.W    (A4),D4             ;
          BSR.S     @IMOVEP66           ;
@CS20:    BRA       @COMMON4            ;
@IMOVEP35:BSR.S     @IMOVEP66           ;0XHHHH(A@),D@
          MOVE.B    D5,(A6)+            ;COMMA SEPARATOR
          MOVE.B    (A4),D4             ;
          LSR.B     #1,D4               ;
          BSR       @FORMREGD           ;
          BRA       @CS20               ;COMMON4
@IMOVEP66:MOVE.B    #'$',(A6)+          ;FORMAT DISPLACEMENT
          MOVE.W    2(A4),D0            ;
          BSR       @PNT4HX             ;
          MOVE.B    #'(',(A6)+          ;
          MOVE.W    (A4),D4             ;
          BSR       @FORMREGA           ;
          MOVE.B    #')',(A6)+          ;
```

```
          RTS                              ;
@SCOMMON: BRA       @COMMON                ;NOP RESET RTE RTR RTS TRAPV
@SCC:     BSR       @ICCCC                 ;GET REST OF OP-CODE
          BSR       @EA                    ;DATA ALTERABLE
          BRA       @SCOMMON               ;
@IDBCC:   MOVE.W    (A4),D4                ;DB--
          BSR       @FORMREGD              ;
          MOVE.B    D5,(A6)+               ;COMMA SEPARATOR
          MOVE.B    #'$',(A6)+             ;HEX FIELD TO FOLLOW
          BSR       @ICCCC                 ;
          BRA.S     @ICC55                 ;

/*        BIT  5432109876543210
          0110............                 FIXED
          ....CCCC........                 CONDITION
          ........DDDDDDD0                 DISPLACEMENT
          ...............1                 ERROR (ODD BOUNDRY DISPLACEMENT)
*/
@ICC:     BSR       @ICCCC                 ;B--
@IBSR:    MOVE.B    #'$',(A6)+             ;BSR   BRA
          TST.B     D4                     ;
          BEQ.S     @ICC55                 ;16 BIT DISPLACEMENT
          MOVE.B    #'.',(A5)+             ;
          MOVE.B    #'S',(A5)+             ;
          EXT.W     D4                     ;8 BIT DISPLACEMENT
@ICC35:   EXT.L     D4                     ;SIGN-EXTENDED DISPLACEMENT
          ADD.L     A2,D4                  ;+ PROGRAM COUNTER
          ADD.L     #2,D4                  ;+ TWO
          MOVE.L    D4,D0                  ;
          ASR.L     #1,D4                  ;
          BCS       @FERROR                ;ODD BOUNDRY DISPLACEMENT
          BSR       @PNT6HX                ;
          BRA       @SCOMMON               ;
@ICC55:   ADD.L     #2,D3                  ;SIZE
          MOVE.W    2(A4),D4               ;
          MOVE.B    #'.',(A5)+             ;
          MOVE.B    #'L',(A5)+             ;.L FOR 16 BIT DISPLACEMENT
          BRA       @ICC35                 ;

/*        BCHG  BCLR  BSET  BTST           */
@ISETD:   ROL.W     #7,D4                  ;DYNAMIC BIT
          BSR       @FORMREGD              ;DATA REGISTER
@ISETD12: MOVE.B    D5,(A6)+               ;COMMA SEPARATOR
          MOVE.W    (A4),D4                ;
          BSR       @EA                    ;DATA ALTERABLE
@CS18:    BRA       @SCOMMON

/*        BCHG  BCLR  BSET  BTST
          1ST WORD  .... .... ..XX XXXX    EA   DATA ALTERABLE ONLY
          2ND WORD  0000 0000 000Y YYYY    BIT NUMBER
```

```
*/
@ISETS:    ADD.L     #2,D3              ;STATIC BIT, SIZE
           MOVE.B    #'#',(A6)+         ;IMMEDIATE
           CLR.L     D0                 ;
           MOVE.W    2(A4),D0           ;GET BIT POSITION FROM 2ND WORD
           MOVE.L    D0,D1              ;
           LSR.L     #5,D1              ;
           BNE       @FERROR            ;
           BSR       @HEX2DEC           ;DECIMAL
           BRA       @ISETD12           ;

/*         BIT  5432109876543210
           ....XXX.........           IMMEDIATE COUNT/REGISTER
           .......0........           RIGHT SHIFT
           .......1........           LEFT SHIFT
           ........00......           BYTE
           ........01......           WORD
           ........10......           LONG
           ....0...11......           WORD (MEMORY)
           ....0...11AAAAAA           EFFECTIVE ADDRESS
           ..........0.....           SHIFT IMMEDIATE COUNT
           ..........1.....           SHIFT COUNT (MODULO 64) IN DATA REG
*/
@ISHIFT:   MOVE.W    #(0X4C52),D0       ;'LR' AS-  LS-  RO-  ROX-
           BTST      #8,D4              ;DIRECTION BIT
           BEQ.S     @ISHIFT13          ;RIGHT
           LSR.W     #8,D0              ;LEFT
@ISHIFT13:MOVE.B     D0,(A5)+           ;DIRECTION "L" OR "R"
           MOVE.W    (A4),D0            ;
           AND.W     #0X00C0,D0         ;
           CMP.W     #0X00C0,D0         ;
           BEQ.S     @ISHIFTM1          ;MEMORY SHIFT
           BSR       @FORMSIZE          ;
           ROL.W     #7,D4              ;
           BTST      #12,D4             ;I/R BIT
           BNE.S     @ISHIFT33          ;COUNT IN REGISTER
           AND.B     #0X07,D4           ;IMMEDIATE COUNT
           BNE.S     @ISHIFT23          ;
           OR.B      #0X08,D4           ;CHANGE ZERO TO EIGHT
@ISHIFT23:OR.B       #'0',D4            ;
           MOVE.B    #'#',(A6)+         ;
           MOVE.B    D4,(A6)+           ;
           BRA.S     @ISHIFT44          ;
@ISHIFT33:BSR        @FORMREGD          ;
@ISHIFT44:MOVE.B     D5,(A6)+           ;COMMA SEPARATOR
           MOVE.W    (A4),D4            ;
           BSR       @FORMREGD          ;
@CS17:     BRA       @CS18              ;COMMON
@ISHIFTM1:MOVE.B     #'.',(A5)+         ;PERIOD
           MOVE.B    #'W',(A5)+         ;.WORD
```

```
        BTST    #11,D4              ;
        BNE     @FERROR             ;BIT 11 MUST BE ZERO
        MOVE.W  #0X1FC,D7           ;MEMORY ALTERABLE ADDRESSING
        BSR     @EA                 ;
        BRA     @CS17               ;COMMON
@ICCCC: MOVE.L  #0X0F,D0            ;APPEND CONDITION CODE          1,4
        AND.B   (A4),D0             ;D0 = CCC                       1,4
        LSL.L   #1,D0               ;D0 = CCC*2
        MOVE.L  A0,-(SP)            ;
        LEA     @BRTBL,A0           ;
        ADD.L   D0,A0               ;
        MOVE.B  (A0),D1             ;
        LSL.L   #4,D1               ;
        LSL.L   #4,D1               ;
        MOVE.B  1(A0),D1            ;
        MOVE.L  (SP)+,A0            ;
        MOVE.W  @BRTBL(PC,D0.W),D1  ;GET BRANCH MNEMONIC
        MOVE.B  D1,(A5)+            ;(REVERSED) FROM THE TABLE
        LSR.W   #8,D1               ;AND ADD THE NONBLANK PORTION
        CMP.B   #' ',D1             ;TO THE BUFFER.
        BEQ.S   @ICCCC9             ;
        MOVE.B  D1,(A5)+            ;
@ICCCC9: RTS                        ;

@BRTBL: DC.B    ' ','T'             ;'T',' '  BRA ACCEPTED
        DC.B    ' ','F' ;'F',' '    ;
        DC.B    'I','H' ;'H','I'    ;
        DC.B    'S','L' ;'L','S'    ;
        DC.B    'C','C' ;'C','C'    ;
        DC.B    'S','C' ;'C','S'    ;
        DC.B    'E','N' ;'N','E'    ;
        DC.B    'Q','E' ;'E','Q'    ;
        DC.B    'C','V' ;'V','C'    ;
        DC.B    'S','V' ;'V','S'    ;
        DC.B    'L','P' ;'P','L'    ;
        DC.B    'I','M' ;'M','I'    ;
        DC.B    'E','G' ;'G','E'    ;
        DC.B    'T','L' ;'L','T'    ;
        DC.B    'T','G' ;'G','T'    ;
        DC.B    'E','L' ;'L','E'    ;
'*      BIT  5432109876543210
        ....RRRMMM......            DESTINATION REGISTER MODE
        ..........MMMRRR            SOURCE MODE REGISTER

        IF BYTE SIZE, ADDRESS DIRECT NOT ALLOWED AS SOURCE
'/
!IMOVEA1: MOVE.W  #0XFFF,D7         ;ALL MODES
        BSR     @EA                 ;
        MOVE.B  D5,(A6)+            ;COMMA SEPARATOR
        MOVE.W  (A4),D4             ;....RRRMMM......
```

```
          LSR.W     #1,D4               ;.....RRRMMM.....
          LSR.B     #5,D4               ;.....RRR.....MMM
          ROR.W     #8,D4               ;.....MMM.....RRR
          LSL.B     #5,D4               ;.....MMMRRR.....
          LSR.W     #5,D4               ;..........MMMRRR

/*        IF .BYTE DESTINATION A@ NOT ALLOWED  */
          MOVE.W    #0X1FF,D7           ;DATA ALTERABLE + A@
          MOVE.B    (A4),D0             ;
          CMP.B     #0X01,D0            ;
          BNE.S     @IMOVE19            ;NOT BYTE SIZE
          MOVE.W    #0X1FD,D7           ;DATA ALTERABLE
@IMOVE19: BSR       @EA                 ;
          BRA.S     @CS19               ;COMMON

/*        IF BYTE,  ADDRESS REGISTER DIRECT NOT ALLOWED  */
@IQUICKA: BSR.S     @FORMSIZE           ;ADDQ  SUBQ
          MOVE.B    #'#',(A6)+          ;
          ROL.W     #7,D4               ;
          AND.B     #7,D4               ;
          BNE.S     @IQUICK21           ;
          OR.B      #8,D4               ;MAKE ZERO INTO EIGHT
@IQUICK21:OR.B      #'0',D4             ;MAKE ASCII
          MOVE.B    D4,(A6)+            ;
          MOVE.B    D5,(A6)+            ;COMMA SEPARATOR
          MOVE.W    (A4),D4             ;
          MOVE.W    (A4),D0             ;
          AND.W     #0X00C0,D0          ;
          BEQ.S     @IQUICK31           ;DATA ALTERABLE
          MOVE.W    #0X1FF,D7           ;ALTERABLE ADDRESSING
@IQUICK31:BSR       @EA                 ;
@CS19:    BRA       @COMMON             ;

/*        BIT   5432109876543210
          ........00......              BYTE
          ........01......              WORD
          ........10......              LONG
          ........11......              ERROR
*/
@FORMSIZE:MOVE.W    (A4),D2             ;
          MOVE.B    #'.',(A5)+          ;STORE PERIOD
          LSR.W     #6,D2               ;
          AND.W     #0X03,D2            ;
          BNE.S     @FORM91             ;
          MOVE.B    #'B',(A5)+          ;STORE "B"
          BRA.S     @FORM95             ;
@FORM91:  MOVE.B    #'W',D0             ;
          CMP.B     #1,D2               ;
          BEQ.S     @FORM93             ;
          MOVE.B    #'L',D0             ;
```

```
          CMP.B     #2,D2               ;
          BNE.S     @FE10               ;FERROR
@FORM93:  MOVE.B    D0,(A5)+            ;STORE "W" OR "L"
@FORM95:  RTS                         ;
@EA000:   BSR       @FORMREGD           ;
          BTST      #0,D7               ;
          BEQ.S     @FE10               ;FERROR
          RTS                         ;
@EA001:   BSR       @FORMREGA           ;
          BTST      #1,D7               ;
          BEQ.S     @FE10               ;FERROR     THIS MODE NOT ALLOWED
          RTS                         ;
@EA010:   MOVE.B    #'(',(A6)+          ;
          BSR       @FORMREGA           ;
          MOVE.B    #')',(A6)+          ;
          BTST      #2,D7               ;
          BEQ.S     @FE10               ;FERROR     THIS MODE NOT ALLOWED
          RTS                         ;
@EA011:   MOVE.B    #'(',(A6)+          ;
          BSR       @FORMREGA           ;
          MOVE.B    #')',(A6)+          ;
          MOVE.B    #'+',(A6)+          ;
          BTST      #3,D7               ;
          BEQ.S     @FE10               ;FERROR     THIS MODE NOT ALLOWED
@EA011RTS:RTS                         ;
@EA100:   MOVE.B    #'-',(A6)+          ;
          MOVE.B    #'(',(A6)+          ;
          BSR       @FORMREGA           ;
          MOVE.B    #')',(A6)+          ;
          BTST      #4,D7               ;
          BNE       @EA011RTS           ;
@FE10:    BRA       @FERROR             ;THIS MODE NOT ALLOWED

/*        A4 = POINTER TO FIRST WORD
          D3 = OFFSET TO EXTENSION
          D4 = VALUE TO PROCESS
          D7 = MODES ALLOWED MASK
*/
@EA:      MOVE.L    D4,D0               ;
          LSR.W     #3,D0               ;
          AND.W     #0X7,D0             ;
          BEQ       @EA000              ;
          CMP.B     #1,D0               ;
          BEQ       @EA001              ;
          CMP.B     #2,D0               ;
          BEQ       @EA010              ;
          CMP.B     #3,D0               ;
          BEQ       @EA011              ;
          CMP.B     #4,D0               ;
          BEQ       @EA100              ;
```

```
            CMP.B     #5,D0             ;
            BEQ.S     @EA101            ;
            CMP.B     #7,D0             ;
            BEQ       @EA111            ;

/*          EXTENSION WORD              */
/*          BIT  5432109876543210
            0...............            DATA REGISTER
            1...............            ADDRESS REGISTER
            .RRR............            REGISTER
            ....0...........            SIGN EXT., LOW ORDER INT. IN INDEX R
            ....1...........            LONG VALUE IN INDEX REGISTER
            .....000........
            ........DDDDDDDD            DISPLACEMENT INTEGER
            EA110                       ADDRESS REGISTER INDIRECT WITH INDE
*/
            BTST      #6,D7             ;
            BEQ       @FE10             ;FERROR     THIS MODE NOT ALLOWED
            MOVE.B    0(A4,D3),D1       ;
            LSL.L     #4,D1             ;
            LSL.L     #4,D1             ;
            MOVE.B    1(A4,D3),D1       ;
            AND.W     #0X0700,D1        ;
            BNE       @FE10             ;FERROR    BITS 10-8 MUST BE ZERO
            MOVE.B    0(A4,D3),D0       ;D0 = DISPLACEMENT
            LSL.L     #4,D0             ;
            LSL.L     #4,D0             ;
            MOVE.B    1(A4,D3),D0       ;
            EXT.W     D0                ;
            EXT.L     D0                ;
            BSR       @HEX2DEC          ;DECIMAL
            MOVE.B    #'(',(A6)+        ;(
            BSR       @FORMREGA         ;XX(A@
            MOVE.B    #',',(A6)+        ;XX(A@,
            MOVE.B    0(A4,D3),D4       ;
            ASR.B     #4,D4             ;
            BPL.S     @EA1105           ;
            BSR       @FORMREGA         ;
            BRA.S     @EA1107           ;
@EA1105:    BSR       @FORMREGD         ;
@EA1107:    MOVE.B    #'.',(A6)+        ;XX(A@,X@.
            MOVE.B    0(A4,D3),D4       ;D4 = R@
            LSL.L     #4,D4             ;
            LSL.L     #4,D4             ;
            MOVE.B    1(A4,D3),D4       ;
            MOVE.B    #'W',D0           ;..........W
            BTST      #11,D4            ;
            BEQ.S     @EA1109           ;
            MOVE.B    #'L',D0           ;..........L
@EA1109:    MOVE.B    D0,(A6)+          ;
```

```
        MOVE.B    #')',(A6)+          ;...........)
        ADD.L     #2,D3               ;
        RTS                           ;

/*      ADDRESS REGISTER INDIRECT WITH DISPLACEMENT  */
@EA101: BTST      #5,D7               ;101000   DIS(A@)
        BEQ.S     @FE11               ;FERROR  THIS MODE NOT ALLOWED
        MOVE.B    0(A4,D3),D0         ;
        LSL.L     #4,D0               ;
        LSL.L     #4,D0               ;
        MOVE.B    1(A4,D3),D0         ;
        EXT.L     D0                  ;
        BSR       @HEX2DEC            ;DECIMAL
        ADD.L     #2,D3               ;SIZE
        BRA       @EA010              ;

*       111000                        ABSOLUTE SHORT
        111001                        ABSOLUTE LONG
        111010                        PROGRAM COUNTER WITH DISPLACEMENT
        111011                        PROGRAM COUNTER WITH INDEX
        111100                        IMMEDIATE OR STATUS REG
/
@EA111: AND.W     #7,D4               ;
        BNE.S     @EA1112             ;
        BTST      #7,D7               ;
        BEQ.S     @FE11               ;FERROR  THIS MODE NOT ALLOWED
        MOVE.B    0(A4,D3),D0         ;111000   ABSOLUTE SHORT
        LSL.L     #4,D0               ;
        LSL.L     #4,D0               ;
        MOVE.B    1(A4,D3),D0         ;
        EXT.L     D0                  ;
        MOVE.B    #'$',(A6)+          ;
        BSR       @PNT8HX             ;SIGN EXTENDED VALUE           1,3
        ADD.L     #2,D3               ;SIZE + 2
        RTS                           ;
@EA1112: CMP.B    #1,D4               ;
        BNE.S     @EA1113             ;
        BTST      #8,D7               ;
        BEQ.S     @FE11               ;FERROR  THIS MODE NOT ALLOWED
        MOVE.B    #'$',(A6)+          ;HEX
        MOVE.L    0(A4,D3),D0         ;111001  ABSOLUTE LONG
        BSR       @PNT8HX             ;
*       - MOVE.B    #'.',(A6)+        ;FORCE LONG @FORMAT 1,3 */
*       - MOVE.B    #'L',(A6)+        ;IE    .L 1,3   */
        ADD.L     #4,D3               ;
        RTS                           ;
@EA1113: CMP.B    #2,D4               ;
        BNE.S     @EA1114             ;
        BTST      #9,D7               ;
        BNE.S     @EA1113A            ;
```

```
@FE11:     BRA       @FERROR             ;THIS MODE NOT ALLOWED
@EA1113A:  MOVE.B    0(A4,D3),D0         ;111010 PC+DISPLACEMENT DESTINATION
           LSL.L     #4,D0               ;
           LSL.L     #4,D0               ;
           MOVE.B    1(A4,D3),D0         ;
           EXT.L     D0                  ;
           ADD.L     A2,D0               ;
           ADD.L     #2,D0               ;
           MOVE.B    #'$',(A6)+          ;HEX "$"          1,3
           BSR       @PNT8HX             ;DESTINATION
           MOVE.L    #(0X29435028),D0    ;(PC) ')CP('
           BSR       @SCHR               ;STORE WORD
           ADD.L     #2,D3               ;SIZE
           RTS                           ;
@EA1114:   CMP.B     #3,D4               ;
           BNE     @EA1115               ;
                                         ;
/*         PROGRAM COUNTER WITH INDEX    DESTINATION(PC,R@.X)   */
/*         5432109876543210              SECOND WORD
           0...............              DATA REGISTER
           1...............              ADDRESS REGISTER
           .XXX............              REGISTER
           ....0...........              SIGN-EXTENDED, LOW ORDER WORD INTEG
           ..IN INDEX REGISTER
           ....1...........              LONG VALUE IN INDEX REGISTER
           .....000........
           ........XXXXXXXX              DISPLACEMENT INTEGER
*/
           BTST      #10,D7              ;
           BEQ       @FE11               ;FERROR    THIS MODE NOT ASLLOWED
           MOVE.B    0(A4,D3),D1         ;
           LSL.L     #4,D1               ;
           LSL.L     #4,D1               ;
           MOVE.B    1(A4,D3),D1         ;
           AND.W     #0X0700,D1          ;
           BNE       @FE11               ;FERROR  BITS 10-8 MUST BE ZERO
           MOVE.B    1(A4,D3),D0         ;111100   DESTINATION(PC,R@.X)
           EXT.W     D0                  ;
           EXT.L     D0                  ;
           ADD.L     A2,D0               ;
           ADD.L     #2,D0               ;
           MOVE.B    #'$',(A6)+          ;HEX "$"          1,3
           BSR       @PNT8HX             ;DESTINATION      1,3
           MOVE.L    #(0X2C435028),D0    ;',CP('
           BSR       @SCHR               ;DES(PC,
           MOVE.B    0(A4,D3),D4         ;
           LSL.L     #4,D4               ;
           LSL.L     #4,D4               ;
           MOVE.B    1(A4,D3),D4         ;
           ROL.W     #4,D4               ;
```

```
        BTST      #3,D4              ;
        BEQ.S     @EAF25             ;
        BSR       @FORMREGA          ;
        BRA.S     @EAF27             ;
@EAF25: BSR       @FORMREGD          ;DES(PC,R@
@EAF27: MOVE.B    #'.',(A6)+         ;DES(PC,R@.
        MOVE.B    0(A4,D3),D4        ;
        LSL.L     #4,D4              ;
        LSL.L     #4,D4              ;
        MOVE.B    1(A4,D3),D4        ;
        MOVE.W    #(0X4C57),D0       ;'LW'
        BTST      #11,D4             ;
        BEQ.S     @EAF35             ;
        LSR.W     #8,D0              ;
@EAF35: MOVE.B    D0,(A6)+           ;DES(PC,R@.X
        MOVE.B    #')',(A6)+         ;DES(PC,R@.X)
        ADD.L     #2,D3              ;
        RTS                          ;


*        BIT  5432109876543210
        ..........111100             FIRST WORD  #<IMMEDIATE>
/
EA1115: CMP.B     #4,D4              ;
        BNE       @FE11              ;FERROR
        BTST      #11,D7             ;
        BEQ       @FE11              ;FERROR THIS MODE NOT ALLOWED
        MOVE.B    #'#',(A6)+         ;IMMEDIATE
        MOVE.B    -1(A5),D1          ;
        CMP.B     #'L',D1            ;
        BEQ.S     @EA11155           ;LONG
        MOVE.B    0(A4,D3),D0        ;
        LSL.L     #4,D0              ;
        LSL.L     #4,D0              ;
        MOVE.B    1(A4,D3),D0        ;
        CMP.B     #'B',D1            ;
        BNE.S     @EA11153           ;.WORD

*       BYTE SIZE, DATA ALLOWED
        0000 0000 XXXX XXXX
        1111 1111 1XXX XXXX
/
        MOVE.L    D0,D1              ;
        LSR.W     #8,D1              ;
        BEQ.S     @EA11153           ;
        MOVE.L    D0,D1              ;
        ASR.W     #7,D1              ;
        ADD.W     #1,D1              ;
        BNE       @FE11              ;FERROR
```

```
@EA11153: EXT.L     D0                  ;
          BSR       @HEX2DEC            ;
          ADD.L     #2,D3               ;
          RTS                           ;

@EA11155: MOVE.L    0(A4,D3),D0         ;
          BSR       @HEX2DEC            ;
          ADD.L     #4,D3               ;SIZE
          RTS                           ;

@MOVEMS:  MOVE.B    #'.',(A5)+          ;PERIOD
          MOVE.W    #(0X4C57),D0        ;'LW'
          BTST      #6,D4               ;
          BEQ.S     @MOVEMS2            ;
          LSR.W     #8,D0               ;
@MOVEMS2: MOVE.B    D0,(A5)+            ;SIZE
          RTS                           ;

/*        MOVEM -   REGISTER EXPANSION */
@MOVEMR:  MOVE.W    2(A4),D2            ;D2 = SECOND WORD
          MOVE.L    #' ',D0             ;D0 = SPACE
          MOVE.L    #'/',D7             ;D7 = /
          SUB.L     #1,A6               ;ADJUST STORE POINTER
          MOVE.L    #'0',D5             ;D5 = REGISTER #
          MOVE.W    #(0X4144),D4        ;D4 = REG CLASS    'AD'
@MOVEMR11:BTST      D1,D2               ;
          BEQ.S     @MOVEMR77           ;BIT RESET
          CMP.B     (A6),D0             ;BIT SET
          BNE.S     @MOVEMR44           ;NOT SPACE
@MOVEMR33:MOVE.B    D4,1(A6)            ;REG TYPE
          MOVE.B    D5,2(A6)            ;REG #
          MOVE.B    #'-',3(A6)          ;-
          ADD.L     #3,A6               ;
          BRA.S     @MOVEMR88           ;
@MOVEMR44:CMP.B     #',',(A6)           ;
          BEQ       @MOVEMR33           ;COMMA SEPARATOR
          CMP.B     (A6),D7             ;/ SEPARATOR
          BEQ       @MOVEMR33           ;
          MOVE.B    D4,1(A6)            ;REG TYPE
          MOVE.B    D5,2(A6)            ;REG #
          MOVE.B    #'-',3(A6)          ; - SEPARATOR
          BRA.S     @MOVEMR88           ;
@MOVEMR77:CMP.B     #',',(A6)           ;
          BEQ.S     @MOVEMR88           ;COMMA
          CMP.B     (A6),D0             ;
          BEQ.S     @MOVEMR88           ;SPACE
          CMP.B     1(A6),D0            ;
          BEQ.S     @MOVEMR79           ;SPACE
          ADD.L     #3,A6               ;
@MOVEMR79:MOVE.B    D7,(A6)             ;/ SEPARATOR
```

```
@MOVEMR88:ADD.L     #1,D5               ;
          ADD.L     D6,D1               ;D1 = BIT POSITION
          CMP.B     #'8',D5             ;
          BNE       @MOVEMR11           ;
          CMP.B     (A6),D0             ;SPACE
          BEQ.S     @MOVEMR94           ;
          CMP.B     1(A6),D0            ;SPACE
          BEQ.S     @MOVEMR94           ;
          ADD.L     #3,A6               ;
          MOVE.B    D7,(A6)             ;/   SEPARATOR
@MOVEMR94:MOVE.B    #'0',D5             ;RESET REG TO ZERO
          LSR.W     #8,D4               ;CHANGE REG TYPE
          BNE       @MOVEMR11           ;MORE
          MOVE.B    D0,(A6)             ;SPACE
          RTS                           ;
@DECODE:  MOVE.L    20(SP),A5           ;
          MOVE.L    16(SP),D0           ;
          MOVE.L    12(SP),D1           ;
          MOVE.L    8(SP),D2            ;
          MOVE.L    4(SP),A2            ;
          LINK      A1,#-16             ;CREATE A FRAME FOR THE
          MOVEM.L   D0-D2/A4,-16(A1)    ;CODE AND ITS PC.  A4
          LEA       -16(A1),A4          ;POINTS TO THE CODE.
          MOVE.L    A5,A3               ;A3 = START OF OUTPUT BUFFER
          MOVE.L    #80,D0              ;
          MOVE.L    A3,A6               ;
@DEC311:  MOVE.B    #' ',(A6)+          ;SPACE FILL BUFFER
          SUB.L     #1,D0               ;
          BNE       @DEC311             ;

'*        CHECK FOR KNOWN ILLEGAL CODES  */
          MOVE.W    (A4),D0             ;
          LEA       @KI,A5              ;
          MOVE.L    A5,A6               ;
          ADD.L     #2,A6               ;#(@KIEND-@KI)=2
@DEC404:  CMP.W     (A5)+,D0            ;
          BEQ.S     @FE12               ;FERROR  ILLEGAL CODE          1,4
          CMP.L     A6,A5               ;
          BNE       @DEC404             ;
                                        ;
'*        LOOK FOR MATCH OF OP-CODE     */
          MOVEM.L   D1-D2,-(SP)         ;SAVE D1,D2
          MOVE.L    #8,D2               ;8=SHIFT CNT
          LEA       @TBL,A5             ;A5 = POINTER TO DECODE TABLE
          LEA       @TBLE,A6            ;A6 = POINTER TO END OF TABLE
!DEC411:  MOVE.B    (A4),D0             ;FIRST BYTE
          LSL.L     D2,D0               ;
          MOVE.B    1(A4),D0            ;FIRST WORD
          MOVE.B    (A5)+,D1            ;FIRST BYTE
          LSL.L     D2,D1               ;
```

```
            MOVE.B    (A5)+,D1          ;FIRST WORD
            AND.W     D1,D0             ;MASK
            MOVE.B    (A5)+,D1          ;FIRST BYTE
            LSL.L     D2,D1             ;
            MOVE.B    (A5)+,D1          ;FIRST WORD
            CMP.W     D1,D0             ;
            BEQ.S     @DEC425           ;FOUND MATCH
            ADD.L     #4,A5             ;UPDATE POINTER
            CMP.L     A6,A5             ;
            BNE       @DEC411           ;MORE TABLE
            MOVEM.L   (SP)+,D1-D2       ;RESTORE D1,D2
@FE12:      BRA       @FERROR           ;ILLEGAL INSTRUCTION          1,4
@DEC425:    MOVEM.L   (SP)+,D1-D2       ;RESTORE D1,D2
            CLR.L     D6                ;
            MOVE.B    (A5)+,D6          ;D6 = (GOTO OFFSET)/4 ILK BYTE
            LSL.L     #4,D6             ;
            LSL.L     #4,D6             ;
            MOVE.B    (A5)+,D6          ;D6 = (GOTO OFFSET)/4 2ND BYTE
            CLR.L     D7                ;
            MOVE.B    (A5)+,D7          ;D7 = INDEX TO OP-CODE
            ADD.L     #1,A5             ;

/*          MOVE OP-CODE TO BUFFER      */
            LEA       @OPCTBL,A0        ;
@DEC510:    TST       D7                ;
            BEQ.S     @DEC530           ;AT INDEX
@DEC515:    TST.B     (A0)+             ;
            BPL       @DEC515           ;MOVE THROUGH FIELD
            SUB.L     #1,D7             ;
            BRA       @DEC510           ;
@DEC530:    MOVE.L    #30,D0            ;.1,4
            LEA.L     0(A3,D0),A5       ;A5 = STORE POINTER  OP-CODE   1,4
@DEC535:    MOVE.B    (A0)+,D0          ;
            BCLR      #7,D0             ;
            BNE.S     @DEC537           ;END OF MOVE
            MOVE.B    D0,(A5)+          ;
            BRA       @DEC535           ;
@DEC537:    MOVE.B    D0,(A5)+          ;
/*          CALCULATE GOTO AND GO       */
            MOVE.L    #2,D3             ;D3= SIZE
            LEA       @PGM,A0;          ;
            ADD.L     D6,A0;            ;
            MOVE.L    #39,D0            ;1,4
            LEA.L     0(A3,D0),A6       ;A6 = POINTER FOR OPERAND     1,4
            MOVE.W    (A4),D4           ;D4 = FIRST WORD
            MOVE.B    #',',D5           ;D5 = CONTAINS ASCII COMMA
            MOVE.W    #0X1FD,D7         ;D7 = DATA ALTERABLE MODES ALLOWED
            JMP       (A0)              ;

/*          A4 = POINTER TO DATA IN FRAME CREATED BY 'LINK A1,...'
```

```
          A5 = POINTER STORE OP-CODE
          A6 = POINTER STORE OPERAND
          D3 = SIZE = 2 BYTES
          D4 = FIRST WORD
          D7 = ADDRESS MODES ALLOWED (0X1FD) DATA ALTERABLE
*/
@COMMON4: ADD.L     #2,D3               ;SIZE = 4
@COMMON:  MOVE.L    D3,D6               ;D6 = SIZE
          MOVE.B    #' ',(A6)+          ;SPACE AS LAST CHAR
          MOVE.L    A6,A5               ;SAVE END OF BUFFER POINTER
          MOVE.L    #3,D0               ;1,4
          LEA.L     0(A3,D0),A6         ;1,4
@COMMON35:MOVE.W    (A4)+,D0            ;GET NEXT WORD OF DATA.
          ADD.L     #2,A2               ;ADJUST PROG COUNTER.
          BSR       @PNT4HX             ;FORMAT DATA. (A6)+
          SUB.B     #2,D3;              ;
          BNE       @COMMON35;          ;
          MOVE.L    A5,A6               ;A6 = RESTORE END POINTER
          MOVE.L    A3,A5               ;A5 =  BEGINNING OF BUFFER
          MOVE.L    A2,A4               ;MOVE THE UPDATED PC
          UNLK      A1                  ;TO A4 AND UNDO FRAME.
          MOVE.L    A2,4(SP)            ;
          RTS                           ;

/*        ILLEGAL   INSTRUCTION         */
@FERROR:  MOVE.L    #30,D0              ;.1,4
          LEA.L     0(A3,D0),A6         ;1,4
          LEA       @MSG111,A5          ;
@FERROR35:MOVE.B    (A5)+,D0            ;
          CMP.B     #4,D0               ;
          BEQ.S     @FERROR39           ;
          MOVE.B    D0,(A6)+            ;
          BRA       @FERROR35           ;
@FERROR39:MOVE.W    (A4),D0             ;
          BSR       @PNT4HX             ;
          MOVE.L    #2,D3     ;SIZE     ;
          BRA       @COMMON             ;
@MSG111:  DC.B      'W','O','R','D';    ;
          DC.B      ' ',' ',' ',' ';    ;
          DC.B      '$',4;              ;

@KI:      DC.W       0X4AFB             ;KNOWN ILLEGAL CODES
@KIEND:
     \1   MASK
     \2   OP-CODE PATTERN
     \3   GOTO OFFSET
     \4   INDEX TO OP-CODE

 TBL:     DC.L    0XFEC0E6C0            ;RO
 ISH1:    DC.W    0X0000                ;
```

```
            DC.B    56                  ;
            DC.L    0XFEC0E4C0          ;
@ISH2:      DC.W    0X000               ;
            DC.B    57                  ;ROX
            DC.L    0XFEC0E2C0          ;
@ISH3:      DC.W    0X0000              ;
            DC.B    55                  ;LS
            DC.L    0XFEC0E0C0          ;
@ISH4:      DC.W    0X0000              ;
            DC.B    54                  ;AS
            DC.L    0XF018E018          ;
@ISH5:      DC.W    0X0000              ;
            DC.B    56                  ;RO
            DC.L    0XF018E010          ;
@ISH6:      DC.W    0X0000              ;
            DC.B    57                  ;ROX
            DC.L    0XF018E008          ;
@ISH7:      DC.W    0X0000              ;
            DC.B    55                  ;LS
            DC.L    0XF018E000          ;
@ISH8:      DC.W    0X0000              ;
            DC.B    54                  ;AS
            DC.L    0XF0C0D0C0          ;
@F10EX1:    DC.W    0X0000              ;
            DC.B    4                   ;ADD        <EA> A@
            DC.L    0XF130D100          ;
@F124:      DC.W    0X0000              ;
            DC.B    53                  ;ADDX
            DC.L    0XF000D000          ;
@F10EX3:    DC.W    0X0000              ;
            DC.B    4                   ;ADD
            DC.L    0XF1F8C188          ;
@F91:       DC.W    0X0000              ;
            DC.B    50                  ;EXG
            DC.L    0XF1F8C148          ;
@F81:       DC.W    0X0000              ;
            DC.B    50                  ;EXG
            DC.L    0XF1F8C140          ;
@F71:       DC.W    0X0000              ;
            DC.B    50                  ;EXG
            DC.L    0XF1F0C100          ;
@F121:      DC.W    0X0000              ;
            DC.B    49                  ;ABCD
            DC.L    0XF1C0C1C0          ;
@F6D1:      DC.W    0X0000              ;
            DC.B    48                  ;MULS
            DC.L    0XF1C0C0C0          ;
@F6D2:      DC.W    0X0000              ;
            DC.B    47                  ;MULU
            DC.L    0XF000C000          ;
```

```
@F101:     DC.W    0X0000              ;
           DC.B    2                   ;AND
           DC.L    0XF0C0B0C0          ;
@F10EX4:   DC.W    0X0000              ;
           DC.B    6                   ;CMP      <EA> A@
           DC.L    0XF138B108          ;
@F12A1:    DC.W    0X0000              ;
           DC.B    46                  ;CMPM
           DC.L    0XF100B100          ;
@F102:     DC.W    0X0000              ;
           DC.B    5                   ;EOR
           DC.L    0XF000B000          ;
@F10EX5:   DC.W    0X0000              ;
           DC.B    6                   ;CMP
           DC.L    0XF0C090C0          ;
@F10EX6:   DC.W    0X0000              ;
           DC.B    44                  ;SUB     <EA> A@
           DC.L    0XF1309100          ;
@F122:     DC.W    0X0000              ;
           DC.B    45                  ;SUBX
           DC.L    0XF0009000          ;
@F10EX2:   DC.W    0X0000              ;
           DC.B    44                  ;SUB
           DC.L    0XF1F08100          ;
@F123:     DC.W    0X0000              ;
           DC.B    43                  ;SBCD
           DC.L    0XF1C081C0          ;
@F6D3:     DC.W    0X0000              ;
           DC.B    42                  ;DIVS
           DC.L    0XF1C080C0          ;
@F6D4:     DC.W    0X0000              ;
           DC.B    41                  ;DIVU
           DC.L    0XF0008000          ;
@F103:     DC.W    0X0000              ;
           DC.B    40                  ;OR
           DC.L    0XF1007000          ;
@IMVQ1:    DC.W    0X0000              ;
           DC.B    39                  ;MOVEQ
           DC.L    0XFF006100          ;
@IBSR1:    DC.W    0X0000              ;
           DC.B    51                  ;BSR
           DC.L    0XFF006000          ;
@IBSR2:    DC.W    0X0000              ;
           DC.B    65                  ;BRA      1 3
           DC.L    0XF0006000          ;
@ICC1:     DC.W    0X0000              ;
           DC.B    38                  ;B
           DC.L    0XF0F850C8          ;
@IDBCC1:   DC.W    0X0000              ;
           DC.B    37                  ;DB
```

```
            DC.L    0XF0C050C0          ;
@SCC1:      DC.W    0X0000              ;
            DC.B    36                  ;S
            DC.L    0XF1005100          ;
@IQUICK1:   DC.W    0X0000              ;
            DC.B    35                  ;SUBQ
            DC.L    0XF1005000          ;
@IQUICK2:   DC.W    0X0000              ;
            DC.B    34                  ;ADDQ
            DC.L    0XF1C041C0          ;
@F6A1:      DC.W    0X0000              ;
            DC.B    33                  ;LEA
            DC.L    0XF1C04180          ;
@F6D5:      DC.W    0X0000              ;
            DC.B    32                  ;CHK
            DC.L    0XFFC04EC0          ;
@F11SL1:    DC.W    0X0000              ;
            DC.B    31                  ;JMP      1 4
            DC.L    0XFFC04E80          ;
@F11SL2:    DC.W    0X0000              ;
            DC.B    30                  ;JSR      1,4
            DC.L    0XFFFF4E77          ;
@SCOMMON1:DC.W      0X0000              ;
            DC.B    29                  ;RTR
            DC.L    0XFFFF4E76          ;
@SCOMMON2:DC.W      0X0000              ;
            DC.B    28                  ;TRAPV
            DC.L    0XFFFF4E75          ;
@SCOMMON3:DC.W      0X0000              ;
            DC.B    27                  ;RTS
            DC.L    0XFFFF4E73          ;
@SCOMMON4:DC.W      0X0000              ;
            DC.B    26                  ;RTE
            DC.L    0XFFFF4E72          ;
@ISTOP1:    DC.W    0X0000              ;
            DC.B    25                  ;STOP
            DC.L    0XFFFF4E71          ;
@SCOMMON5:DC.W      0X0000              ;
            DC.B    24                  ;NOP
            DC.L    0XFFFF4E70          ;
@SCOMMON6:DC.W      0X0000              ;
            DC.B    23                  ;RESET
            DC.L    0XFFF84E68          ;
@IMVFUSP1:DC.W      0X0000              ;
            DC.B    60                  ;MOVE FROM USP
            DC.L    0XFFF84E60          ;
@IMVTUSP1:DC.W      0X0000              ;
            DC.B    60                  ;MOVE TO USP
            DC.L    0XFFF84E58          ;
@F51:       DC.W    0X0000              ;
```

```
          DC.B    22                  ;UNLINK
          DC.L    0XFFF84E50          ;
@ILINK1:  DC.W    0X0000              ;
          DC.B    21                  ;LINK
          DC.L    0XFFF04E40          ;
@F41:     DC.W    0X0000              ;
          DC.B    20                  ;TRAP
          DC.L    0XFF804C80          ;
@IMVMTR1: DC.W    0X0000              ;
          DC.B    15                  ;MOVEM FROM REGISTERS
          DC.L    0XFFC04AC0          ;
@F1A1:    DC.W    0X0000              ;
          DC.B    19                  ;TAS
          DC.L    0XFF004A00          ;
@F11:     DC.W    0X0000              ;
          DC.B    18                  ;TST
          DC.L    0XFFF848C0          ;
@F31:     DC.W    0X0000              ;
          DC.B    17                  ;EXT.L
          DC.L    0XFFF84880          ;
@F32:     DC.W    0X0000              ;
          DC.B    16                  ;EXT.W
          DC.L    0XFF804880          ;
@IMVMFR1: DC.W    0X0000              ;
          DC.B    15                  ;MOVEA TO REGISTERS
          DC.L    0XFFF84840          ;
@F33:     DC.W    0X0000              ;
          DC.B    14                  ;SWAP
          DC.L    0XFFC04840          ;
@F111:    DC.W    0X0000              ;
          DC.B    13                  ;PEA
          DC.L    0XFFC04800          ;
@F1A2:    DC.W    0X0000              ;
          DC.B    12                  ;NBCD
          DC.L    0XFFC046C0          ;
@IMVTSR1: DC.W    0X0000              ;
          DC.B    59                  ;MOVE TO SR
          DC.L    0XFF004600          ;
@F12:     DC.W    0X0000              ;
          DC.B    11                  ;NOT
          DC.L    0XFFC044C0          ;
@IMVTCCR1:DC.W    0X0000              ;
          DC.B    59                  ;MOVE TO CCR
          DC.L    0XFF004400          ;
@F13:     DC.W    0X0000              ;
          DC.B    10                  ;NEG
          DC.L    0XFF004200          ;
@F14:     DC.W    0X0000              ;
          DC.B    9                   ;CLR
          DC.L    0XFFC040C0          ;
```

```
@IMVFSR1: DC.W    0X0000              ;
          DC.B    59                  ;MOVE.W  FROM  SR
          DC.L    0XFF004000          ;
@F15:     DC.W    0X0000              ;
          DC.B    8                   ;NEGX
          DC.L    0XF0003000          ;
@IMOVE1:  DC.W    0X0000              ;
          DC.B    59                  ;MOVE.W
          DC.L    0XF0002000          ;
@IMOVE2:  DC.W    0X0000              ;
          DC.B    60                  ;MOVE.L
          DC.L    0XF0001000          ;
@IMOVE3:  DC.W    0X0000              ;
          DC.B    58                  ;MOVE.B
          DC.L    0XFF000C00          ;
@IMMED1:  DC.W    0X0000              ;
          DC.B    6                   ;CMP       #
          DC.L    0XFF000A00          ;
@IMMED2:  DC.W    0X0000              ;
          DC.B    5                   ;EOR       #
          DC.L    0XFF000600          ;
@IMMED3:  DC.W    0X0000              ;
          DC.B    4                   ;ADD       #
          DC.L    0XFF000400          ;
@IMMED4:  DC.W    0X0000              ;
          DC.B    3                   ;SUB       #
          DC.L    0XFF000200          ;
@IMMED5:  DC.W    0X0000              ;
          DC.B    2                   ;AND       #
          DC.L    0XFF000000          ;
@IMMED6:  DC.W    0X0000              ;
          DC.B    1                   ;OR        #
          DC.L    0XF1380108          ;
@IMOVEP1: DC.W    0X000               ;
          DC.B    0                   ;MOVEP
          DC.L    0XFFC008C0          ;
@ISETS1:  DC.W    0X0000              ;
          DC.B    64                  ;BSET
          DC.L    0XFFC00880          ;
@ISETS2:  DC.W    0X0000              ;
          DC.B    63                  ;BCLR
          DC.L    0XFFC00840          ;
@ISETS3:  DC.W    0X0000              ;
          DC.B    62                  ;BCHG
          DC.L    0XFFC00800          ;
@ISETS4:  DC.W    0X0000              ;
          DC.B    61                  ;BTST
          DC.L    0XF1C001C0          ;
@ISETD1:  DC.W    0X0000              ;
          DC.B    64                  ;BSET
```

```
        DC.L    0XF1C00180          ;
@ISETD2:DC.W    0X0000              ;
        DC.B    63                  ;BCLR
        DC.L    0XF1C00140          ;
@ISETD3:DC.W    0X0000              ;
        DC.B    62                  ;BCHG
        DC.L    0XF1C00100          ;
@ISETD4:DC.W    0X0000              ;
        DC.B    61                  ;BTST

@TBLE:
@OPCTBL:
        DC.B    'M','O','V','E'     ;
        DC.B    128+'P','O',128+'R','';
        DC.B    'N',128+'D','S','U' ;
        DC.B    128+'B','A','D',128+'';
        DC.B    'E','O',128+'R','C' ;
        DC.B    'M',128+'P','M','O' ;
        DC.B    'V',128+'E','N','E' ;
        DC.B    'G',128+'X','C','L' ;
        DC.B    128+'R','N','E',128+'G';
        DC.B    'N','O',128+'T','N' ;
        DC.B    'B','C',128+'D','P' ;
        DC.B    'E','A','.',128+'L' ;
        DC.B    'S','W','A','P'     ;
        DC.B    '.',128+'W','M','O' ;
        DC.B    'V','E',128+'M','E' ;
        DC.B    'X','T','.',128+'W' ;
        DC.B    'E','X','T','.'     ;
        DC.B    128+'L','T','S',128+'T';
        DC.B    'T','A','S','.'     ;
        DC.B    128+'B','T','R','A' ;
        DC.B    128+'P','L','I','N' ;
        DC.B    128+'K','U','N','L' ;
        DC.B    128+'K','R','E','S' ;
        DC.B    'E',128+'T','N','O' ;
        DC.B    128+'P','S','T','O' ;
        DC.B    128+'P','R','T',128+'E';
        DC.B    'R','T',128+'S','T' ;
        DC.B    'R','A','P',128+'V' ;
        DC.B    'R','T',128+'R','J' ;
        DC.B    'S',128+'R','J','M' ;
        DC.B    128+'P','C','H','K' ;
        DC.B    '.',128+'W','L','E' ;
        DC.B    'A','.',128+'L','A' ;
        DC.B    'D','D',128+'Q','S' ;
        DC.B    'U','B',128+'Q',128+'S';
        DC.B    'D',128+'B',128+'B','M';
        DC.B    'O','V','E','Q'     ;
        DC.B    '.',128+'L','O',128+'R';
```

```
          DC.B    'D','I','V','U'       ;
          DC.B    '.',128+'W','D','I'   ;
          DC.B    'V','S','.',128+'W'   ;
          DC.B    'S','B','C',128+'D'   ;
          DC.B    'S','U',128+'B','S'   ;
          DC.B    'U','B',128+'X','C'   ;
          DC.B    'M','P',128+'M','M'   ;
          DC.B    'U','L','U','.'       ;
          DC.B    128+'W','M','U','L'   ;
          DC.B    'S','.',128+'W','A'   ;
          DC.B    'B','C',128+'D','E'   ;
          DC.B    'X',128+'G','B','S'   ;
          DC.B    128+'R','N','U','L'   ;
          DC.B    128+'L','A','D','D'   ;
          DC.B    128+'X','A',128+'S','L';
          DC.B    128+'S','R',128+'O','R';
          DC.B    'O',128+'X','M','O'   ;
          DC.B    'V','E','.',128+'B'   ;
          DC.B    'M','O','V','E'       ;
          DC.B    '.',128+'W','M','O'   ;
          DC.B    'V','E','.',128+'L'   ;
          DC.B    'B','T','S',128+'T'   ;
          DC.B    'B','C','H',128+'G'   ;
          DC.B    'B','C','L',128+'R'   ;
          DC.B    'B','S','E',128+'T'   ;
          DC.B    'B','R',128+'A','E'   ;


/*        PRINT HEX ROUTINES                */
/*        PRINT 8 HEX CHARACTERS            */
/*        D0,D1,D2 DESTROYED                */

@PNT8HX:  SWAP      D0                      ;FLIP REG HALVES
          BSR.S     @PNT4HX                 ;DO TOP WORD
          SWAP      D0                      ;NOW DO LOWER WORD
          BRA.S     @PNT4HX                 ;
/*        PRINT     6 HEX CHARACTERS        */
@PNT6HX:  SWAP      D0                      ;FLIP REGISTER HALVES
          BSR.S     @PNT2HX                 ;
          SWAP      D0                      ;FLIP BACK REG HALVES
/*        PRINT4 HEX CHARACTERS IN D0.W     */
@PNT4HX:  MOVE.W    D0,D1                   ;SAVE IN TEMP
          ROR.W     #8,D0                   ;GET BITS 15-8 INTO LOWER BYTE
          BSR.S     @PNT2HX                 ;PRINT IT
          MOVE.W    D1,D0                   ;PULL IT BACK
/*        PRINT     2 HEX CHARACTERS IN D0.B  */
@PNT2HX:  MOVE.W    D0,D2                   ;SAVE IN TEMP REG
          ROXR.W    #4,D0                   ;FORM UPPER NIBBLE
```

```
        BSR.S     @PUTHEX             ;PUT ASCII INTO PRINT BUFFER
        MOVE.W    D2,D0               ;GET BACK FROM TEMP
/*      CONVERT   D0.NIBBLE TO HEX & PUT IT IN PRINT BUFFER  */
@PUTHEX: AND.B    #0X0F,D0            ;SAVE LOWER NIBBLE
        OR.B      #0X30,D0            ;CONVERT TO ASCII
        CMP.B     #0X39,D0            ;SEE IF IT IS>9
        BLE.S     @SAVHEX             ;
        ADD       #7,D0               ;ADD TO MAKE 10=>A
@SAVHEX: MOVE.B   D0,(A6)+            ;PUT IT IN PRINT BUFFER
        RTS                           ;
/*      PRINT HEX (ZERO SURPRESS)     */
@PNTZHX: CLR.L    D4                  ;IS ZERO WHEN SURPRESSING
        MOVE.L    D0,D1               ;SAVE IN TEMP
        BEQ.S     @PNTZ81             ;IF ZERO
        BPL.S     @PNTZ0              ;
        NEG.L     D1                  ;CHANGE TO POSITIVE VALUE
        BMI.S     @PNTZ81             ;WATCH OUT SPECIAL CASE 0X80000000
        MOVE.B    #'-',(A6)+          ;PUT SIGN INTO BUFFER
@PNTZ0:  MOVE.L   #8,D2               ;8 POSSIBLE CHARACTERS
@PNTZ1:  MOVE.L   D1,D0               ;UNSAVE IT
        MOVE.L    D2,D3               ;COUNT DOWN FROM HERE
        SUB.L     #1,D3               ;BACK OFF ONE
        BEQ.S     @PNTZ4              ;IF NO ROTATE SKIP THIS
@PNTZ2:  ASR.L    #4,D0               ;ROTATE LRIGHT
        AND.L     #0XFFFFFFF,D0       ;CLEAR TOP NIBBLE
        SUB.L     #1,D3               ;
        BNE       @PNTZ2              ;
@PNTZ4:  AND.B    #0XF,D0             ;SAVE ONLY NIBBLE
        BNE.S     @PNTZ3              ;
        TST.B     D4                  ;SEE IF STILL SURPRESSING
        BEQ.S     @PNTZ8              ;
@PNTZ3:  BSR      @PUTHEX             ;PUT A HEX CHAR IN BUFFER
        MOVE.B    D0,D4               ;MARK AS NON-SURPRESSING MODE
@PNTZ8:  SUB.L    #1,D2               ;DO ANOTHER CHAR
        BNE       @PNTZ1              ;
        TST.B     D4                  ;SEE IF ANYTHING PRINTED
        BNE.S     @PNTZ9              ;
@PNTZ81: MOVE.B   #'0',(A6)+          ;MOVE AT LEAST ONE ZERO
@PNTZ9:  RTS                          ;
/*      CONVERT BINARY TO DECIMAL  REG D0 PUT IN (A6) BUFFER AS ASCII  */
@HEX2DEC: MOVEM.L D1-D4/D6-D7,-(A7)   ;SAVE REGISTERS
        MOVE.L    D0,D7               ;SAVE IT HERE
        BPL.S     @HX2DC              ;
        NEG.L     D7                  ;CHANGE TO POSITIVE
        BMI.S     @HX2DC57            ;SPECIAL CASE (-0)
        MOVE.B    #'-',(A6)+          ;PUT IN NEG SIGN
@HX2DC:  CLR.W    D4                  ;FOR ZERO SURPRESS
        MOVE.L    #10,D6              ;COUNTER
@HX2DC0: MOVE.L   #1,D2               ;VALUE TO SUB
        MOVE.L    D6,D1               ;COUNTER
```

```
            SUB.L     #1,D1               ;ADJUST - A POWER OF TEN
            BEQ.S     @HX2DC2             ;IF POWER IS ZERO
@HX2DC1:    MOVE.W    D2,D3               ;D3=LOWER WORD
            MULU      #10,D3              ;
            SWAP      D2                  ;D2=UPPER WORD
            MULU      #10,D2              ;
            SWAP      D3                  ;ADD UPPER TO UPPER
            ADD.W     D3,D2               ;
            SWAP      D2                  ;PUT UPPER IN UPPER
            SWAP      D3                  ;PUT LOWER IN LOWER
            MOVE.W    D3,D2               ;D2=UPPER & LOWER
            SUB.L     #1,D1               ;
            BNE       @HX2DC1             ;
@HX2DC2:    CLR.L     D0                  ;HOLDS SUB AMT
@HX2DC22:   CMP.L     D2,D7               ;
            BLT.S     @HX2DC3             ;IF NO MORE SUB POSSIBLE
            ADD.L     #1,D0               ;BUMP SUBS
            SUB.L     D2,D7               ;COUNT DOWN BY POWERS OF TEN
            BRA       @HX2DC22            ;DO MORE
@HX2DC3:    TST.B     D0                  ;ANY VALUE?
            BNE.S     @HX2DC4             ;
            TST.W     D4                  ;ZERO SURPRESS
            BEQ.S     @HX2DC5             ;
@HX2DC4:    ADD.B     #0X30,D0            ;BINARY TO ASCII
            MOVE.B    D0,(A6)+            ;PUT IN BUFFER
            MOVE.B    D0,D4               ;MARK AS NON ZERO SURPRESS
@HX2DC5:    SUB.L     #1,D6               ;NEXT POWER
            BNE       @HX2DC0             ;
            TST.W     D4                  ;SEE IF ANYTHING PRINTED
            BNE.S     @HX2DC6             ;
@HX2DC57:   MOVE.B    #'0',(A6)+          ;PRINT AT LEAST A ZERO
@HX2DC6:    MOVEM.L   (A7)+,D1-D4/D6-D7   ;RESTORE REGISTERS
            RTS

/*          END OF ROUTINE                */
/*          DISASSEMBLY PROGRAM ENDS      */

@IS2:       MOVEM.L (SP)+,D0-D7/A0-A7     ;

}
}
```

v. **Source code of test.c**

```
/* ** test.c ** */

    /* This program determines the start and the  end
       addresses for the download.c program, and also
       contains the user program.                    */

    char *start,*end;

test() {
asm{
    LEA     @1,A0       ;
    MOVE.L  A0,start    ;
    LEA     @2,A0       ;
    MOVE.L  A0,end      ;@2 is already in A0 ...
    LEA     @1,A1       ;
    LEA     @2,A3       ;
    LEA     @3,A2       ;
    SUBA.L  A1,A3       ;
    MOVE.L  A3,D0       ;
    SUB.W   #8,D0       ;
    MOVE.W  D0,(A2)     ;Count of chars to be sent..
    JMP     (A0)        ;
@1: DC.W    0x0000      ;
    DC.W    0x1000      ;download address...
@3: DC.W    0x0000      ;contains count of chars...

/*   *** USER PROGRAM BEGINS ***  */

    .
    ..
    ...
    ....
    ...
    ..
    .


/*   *** USER PROGRAM ENDS ***  */
@2:      NOP        ;
   }
 }
```

B. SOURCE CODE OF MONITOR PROGRAM.

```
;    /* ** ecb.asm ** */
;
;/* These programs reside in the lower addresses of ROM. At the system
;    start-up these routines are copied to RAM. During execution, some of
;    the routines run in ROM and some of them run in RAM.
;    The Main program loops and waits for any command from Macintosh. Afte
;    receiving the command, the program execution is switched to the desir
;    routine. Which in turn, upon its execution returns to Main.
;
;*/
;
/*  Initialization  */
/*
    Function:  1- Copies ROM Contents to RAM.
               2- Autovector Level 4 and Level 6 Interrupt Handler Addres
                  are loaded in their place in Exception Vector Table.
                  Also, other defined Vector entries are filled with the
                  address of STACKFRAME Routine. The purpose is, just to
                  prevent the system from doing undesired things and the
                  loss of system stack space.
               3- The Stack Allocation is done for SSP,ISP,USP.
               4- Makes PHANTOM Low.
*/

ROM        EQU     $00040000
PHAN_LOW   EQU     $00020000
INTR_CHK   EQU     $000E0000
VIOL_VEC   EQU     $00000020
TRAC_VEC   EQU     $00000024
INT1_VEC   EQU     $00000064
INT2_VEC   EQU     $00000068
INT3_VEC   EQU     $0000006C
INT4_VEC   EQU     $00000070
INT5_VEC   EQU     $00000074
INT6_VEC   EQU     $00000078
INT7_VEC   EQU     $0000007C
TRAP_VEC   EQU     $000000BC
WORD_CON   EQU     $0000FFFF
SEND_ZER   EQU     $000C8000
SEND_ONE   EQU     $00040000
TBL_FPCR   EQU     $00000E9C
TBL_FPSR   EQU     $00000EA0
TBL_FIAR   EQU     $00000EA4
TBL_FP0    EQU     $00000EA8
TBL_FP1    EQU     $00000EB4       ;FLOATING POINT REGISTERS.
TBL_FP2    EQU     $00000EC0
```

```
TBL_FP3    EQU     $00000ECC
TBL_FP4    EQU     $00000ED8
TBL_FP5    EQU     $00000EE4
TBL_FP6    EQU     $00000EF0
TBL_FP7    EQU     $00000EFC
TBL_D0     EQU     $00000F08       ;MC68020 REGISTERS.
TBL_D1     EQU     $00000F0C
TBL_D2     EQU     $00000F10
TBL_D3     EQU     $00000F14
TBL_D4     EQU     $00000F18
TBL_D5     EQU     $00000F1C
TBL_D6     EQU     $00000F20
TBL_D7     EQU     $00000F24
TBL_A0     EQU     $00000F28
TBL_A1     EQU     $00000F2C
TBL_A2     EQU     $00000F30
TBL_A3     EQU     $00000F34
TBL_A4     EQU     $00000F38
TBL_A5     EQU     $00000F3C
TBL_A6     EQU     $00000F40
TBL_USP    EQU     $00000F44
TBL_SSP    EQU     $00000F48
TBL_ISP    EQU     $00000F4C
TBL_PC     EQU     $00000F50
TBL_SRHI   EQU     $00000F54       ;HIGH ORDER WORD IS ZERO (ie. $6FEC).
TBL_SR     EQU     $00000F56       ;LOW  ORDER WORD IS SR   (ie. $6FEE).
TBL_VBR    EQU     $00000F58
TBL_CACR   EQU     $00000F5C
TBL_CAAR   EQU     $00000F60
TBL_SFC    EQU     $00000F64
INTR_ENB   EQU     $F8FF           ;DEFINED CONSTANTS.
MASK_7     EQU     $0700
TRAP_15    EQU     $4E4F
MAXINT     EQU     $7FFF
BRKCOUNT   EQU     $0F

/*  Filling Exception Vector Table Entries  */

           ORG     $00000000       ;

           LONG    $00001FFFC      ;INITIAL STACK POINTER (ISP).
           LONG    HERE            ;INITIAL PROGRAM COUNTER.
           LONG    STACKFRAME+ROM  ;VECTOR NUMBER 02
           LONG    STACKFRAME+ROM  ;VECTOR NUMBER 03
           LONG    STACKFRAME+ROM  ;VECTOR NUMBER 04
           LONG    STACKFRAME+ROM  ;VECTOR NUMBER 05
           LONG    STACKFRAME+ROM  ;VECTOR NUMBER 06
           LONG    STACKFRAME+ROM  ;VECTOR NUMBER 07
           LONG    VIOLHANDLER+ROM ;THE ENTRY POINTS TO
                                   ; PRIV.VIOLATION TRACE HANDLER.
```

```
        LONG    TRACEHANDLER+ROM;THE ENTRY POINTS TO TRACE HANDLER.
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 10
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 11
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 12
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 13
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 14
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 15
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 16

        ORG     $00000060       ;

        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 24
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 25
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 26
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 27
        LONG    HANDLER+ROM     ;THE ENTRY POINTS TO INTERRUPT HANDLER.
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 29
        LONG    ABORT+ROM       ;THE ENTRY POINTS TO INTERRUPT HANDLER.
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 31
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 32
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 33
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 34
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 35
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 36
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 37
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 38
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 39
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 40
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 41
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 42
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 43
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 44
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 45
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 46
        LONG    TRAPH+ROM       ;THE ENTRY POINTS TO TRAP #15 HANDLER.
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 48
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 49
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 50
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 51
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 52
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 53
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 54
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 55
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 56
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 57
        LONG    STACKFRAME+ROM  ;VECTOR NUMBER 58

/*  Initializing the Stack Pointers, and making PHANTOM Low. */

        ORG     $00000400       ;THIS ADDRRESS IS THE END OF EXCEPTION
```

```
                                        ; VECTOR TABLE.
HERE:     MOVEA.L #$00,A0               ;LOAD THE START ADDRESS OF PGM. TO RAM...
L1:       MOVE.L  (A0),(A0)+            ;COPY ROM CONTENTS TO RAM.
          CMP.L   #LAST,A0              ;
          BLE     L1                    ;
          NOP                           ;
          MOVE.L  #$00001FC00,D1        ;[D1]<-$1FC00.
          LONG    $4E7B1803             ;[USP]<-[D1].
          MOVE.L  #$1F800,D2            ;[D2]<-$1F800.
          LONG    $4E7B2800             ;[SSP]<-[D1].
          ANDI    #INTR_ENB,SR          ;ENABLE INTERRUPTS.
          MOVE.L  PHAN_LOW,D0           ;MAKE PHANTOM LOW,SWITCH TO RAM.

/*  End of initialization  */

/*  MAIN Routine Below */
/*
    Function:  Loops endlessly, waiting for a command from Macintosh.
               Each command, which is sent by the Macintosh, has a special
               code. These codes for the commands are shown below:

                  '0' for DOWNLOAD   Command.
                  '1' for UPLOAD     Command.
                  '2' for GO         Command.
                  '3' for CALL       Command.
                  '4' for MEMDISPLAY Command.
                  '5' for MEMWRITE   Command.
                  '8' for DOWNLOAD   Command. ( With Coprocessor Enabled ).

               After receiving one of these commands, program execution
               is switched to the desired routine. In case of an error,
               in receiving the command byte, Main simply continues to
               loop, as if no command was received. In this case user may
               retry his last command.
*/
MAIN:     JSR     RUART+ROM             ;GET THE COMMAND IN D3.
          CMP.B   #0,D3                 ;IS IT DOWNLOAD ?..
          BNE.S   SKIP_0                ;IF NOT, CONTINUE TO FIND A MATCH.
          BSR     DOWNLOAD              ;IF YES, DOWNLOAD.
          BRA     MAIN                  ;DOWNLOAD DONE, WAIT FOR THE NEXT COMMAND.
SKIP_0:   CMP.B   #1,D3                 ;IS IT UPLOAD ?..
          BNE.S   SKIP_1                ;IF NOT, CONTINUE TO FIND A MATCH.
          BSR     UPLOAD                ;IF YES, UPLOAD.
          BRA     MAIN                  ;UPLOAD DONE, WAIT FOR THE NEXT COMMAND.
SKIP_1:   CMP.B   #2,D3                 ;IS IT GO ?..
          BNE.S   SKIP_3                ;IF NOT, CONTINUE TO FIND A MATCH.
          BSR     GO                    ;IF YES, GO.
SKIP_3:   CMP.B   #4,D3                 ;IS IT MEMDISPLAY ?..
          BNE.S   SKIP_4                ;IF NOT, CONTINUE TO FIND A MATCH.
          BSR     MEMDISPLAY            ;IF YES, MEMDISPLAY.
```

```
            BRA     MAIN            ;MEMDISPLAY DONE, WAIT FOR THE NEXT COMM
SKIP_4:     CMP.B   #5,D3           ;IS IT MEMWRITE ?..
            BNE.S   SKIP_5          ;IF NOT, CONTINUE TO FIND A MATCH.
            BSR     MEMWRITE        ;IF YES, MEMWRITE.
SKIP_5:     CMP.B   #3,D3           ;IS IT CALL ?..
            BNE.S   SKIP_6          ;IF NOT, CONTINUE TO FIND A MATCH.
            BSR     GO              ;IF YES, GO (IN CASE OF 'CALL' IN MAC.).
SKIP_6:     CMP.B   #8,D3           ;IS IT DOWNLOAD WITH COPROCESSOR ENABLED
            BNE.S   SKIP_7          ;
            BSR     DOWNLOAD        ;
SKIP_7:     BRA     MAIN            ;GO AND WAIT FOR THE NEXT COMMAND.

/*  MAIN Routine Ends  */


/*  Communication Routines (SUART, RUART, DELAY, GETBIT) Below  */


/*  SUART Routine  Below  */
/*
    Function:  SUART sends byte data, which is in low byte of D3.
               Timing is adjusted such that Baud rate of 9600 is obtained.
    Modified
    Registers: D3 is used to pass argument to SUART. Other than that
               register contents are not modified.

    Called by: UPLOAD, FUPLOAD, MEMWRITE, MEMDISPLAY, SCNTS.
*/
SUART:      MOVEM.L D4-D5,-(SP)     ;BITS TO BE SEND SHOULD BE IN D3.
            MOVE.B  #8,D5           ;EIGHT BITS ARE TO BE SENT.
            JSR     SEND_ZER+DELAY1 ;SEND START BIT.
AGAIN:      BTST    #0,D3           ;EVALUATE LEAST SIGNIFICANT BIT.
            BNE.S   ONE             ;
            JSR     SEND_ZER+DELAY1 ;SEND A ZERO .
            JMP     ROM+SKIP        ;
ONE:        JSR     SEND_ONE+DELAY1 ;SEND A ONE.
SKIP:       ROR.B   #1,D3           ;GET THE OTHER BIT.
            SUBQ.B  #1,D5           ;
            BNE.S   AGAIN           ;
            JSR     SEND_ONE+DELAY1 ;SEND FIRST STOP BIT.
            JSR     SEND_ONE+DELAY1 ;SECOND STOP BIT.
            MOVEM.L (SP)+,D4-D5     ;RESTORE ORIGINAL REGISTERS.
            RTS                     ;

/*  SUART Routine  Ends  */


/*  DELAY1 Routine  Below  */
```

```
    Function:  DELAY1 provides a delay of (1/9600) seconds. Which is
               needed to provide a Baud rate of 9600.
    Modified
    Registers: D4 is modified. But this will not affect the original D4
               value, since it was saved in SUART.

    Called by: SUART.
*/
DELAY1:    MOVE.L  #$0A,D4          ;GET THE DELAY LOOP COUNT IN D4.
LOOP1:     NOP                      ;THIS COUNT AND NOPs ASSURE A BIT TIME
           NOP                      ; OF (1/9600) SECONDS.
           SUB.L   #1,D4            ;DELAY1 IS CALLED BY SUART, SINCE IT SAVES
           BGE     LOOP1            ; D4 WE DON'T NEED TO SAVE D4 HERE.
           NOP                      ;
           NOP                      ;
           RTS                      ;RETURN FROM DELAY1 SUBROUTINE

/*  DELAY1 Routine  Ends  */


/*  RUART Routine  Below  */

    Function:  RUART receives a byte of data from RS232 input, at a Baud
               rate of 9600.
    Modified
    Registers: D3 is modified. It is used to pass the received byte to
               the calling routine.

    Called by: MAIN, DOWNLOAD, LDREGTBL, GO, MEMWRITE, MEMDISPLAY, GETLONG.
*/
RUART:     MOVEM.L D0-D2/D4-D7,-(SP);
START:     MOVE.B  #1,D1            ;SET RECEIVE BUFFER ( BIT # 0 ).
           JMP     INTR_CHK+NEXT1   ;CHECK RS232_IN, WAIT FOR THE START BIT
NEXT1:     NOP                      ;
           NOP                      ;ENOUGH  DELAY FOR RS232 INTERRUPTS.
           NOP                      ;
           JMP     ROM+EXIT1        ;
           NOP                      ;
EXIT1:     CMP.B    #0,D1           ;IF [D1]=1, RS232 IN WAS LOW. START BIT
                                    ; CAME.
           BNE.S   START            ;IF [D1]=0, RS232 INPUT WAS HIGH. WAIT FOR
                                    ; THE START BIT.
           MOVE.L  #0,D6            ;
LAB1:      DBF     D6,LAB1          ;
           MOVE.L  $5AB1,D7         ;
           MOVE.B  #1,D1            ;
           JMP     INTR_CHK+NEXT2   ;CHECK RS232_IN.
NEXT2:     NOP                      ;
           NOP                      ;ENOUGH DELAY FOR RS232 INTERRUPT
           NOP                      ;
```

```
            JMP     ROM+EXIT2           ;
            NOP                         ;
EXIT2:      CMP.B   #0,D1               ;IF [D1]=1, RS232 IN LOW. START BIT CAME
            BNE.S   START               ;IF [D1]=0, NO RS232 IN. PREVIOUS START I
            MOVE.L  #7,D6               ; WAS SPURIOUS. WAIT FOR START BIT.
            NOP                         ;
LAB2:       DBF     D6,LAB2             ;START BIT RECEIVED. NOW START TO RECEIV
            MOVE.L  $5AB2,D7            ; FOLLOWING BITS.
            CLR.B   D3                  ;THE BITS WILL BE SHIFTED INTO D3.
            MOVE.B  #8,D2               ;8 IS THE NUMBER OF BITS TO BE RECEIVED.
CIRC:       JSR     ROM+GETBIT          ;GET NEXT BIT.
            OR.B    D1,D3               ;GET THE BIT INTO D3.
            ROR.B   #1,D3               ;
            MOVE.L  $5AB3,D5            ;
            SUB.B   #1,D2               ;
            BNE.S   CIRC                ;AT THE EXIT POINT WE ARE ALREADY ON THE
                                        ; STOP BIT #1. RECEIVING THE BITS ENDS HE
                                        ; NOW CHECK STOP BITS.

            MOVE.B  D3,$5AB4            ;
            JSR     ROM+GETBIT          ;CHECK FIRST STOP BIT.
            BTST    #0,D1               ;
            BNE.S   PASS1               ;IF IT CAME CHECK FOR THE SECOND STOP BI
            JSR     SENDERROR+ROM       ;SEND RECEIVE ERROR TO MACINTOSH.
            MOVEM.L (SP)+,D0-D2/D4-D7; AND
            JMP     MAIN                ; RETURN TO MAIN LOOP, WAIT FOR NEXT
                                        ; COMMAND.
PASS1:      JSR     ROM+GETBIT          ;CHECK SECOND STOP BIT.
            BTST    #0,D1               ;
            BNE.S   PASS2               ;
            JSR     SENDERROR+ROM       ;SEND RECEIVE ERROR TO MACINTOSH..
            MOVEM.L (SP)+,D0-D2/D4-D7;AND...
            JMP     MAIN                ;RETURN TO MAIN LOOP, WAIT FOR NEXT COM
PASS2:      MOVEM.L (SP)+,D0-D2/D4-D7;
            RTS                         ;

/*  RUART Routine  Ends  */


/*  GETBIT Routine  Below  */

    Function:  GETBIT receives a "bit" of data from RS232 input, at a Baud
               rate of 9600. It senses the RS232 input.
    Modified
    Registers: D1 is modified. It is used to pass the received bit to
               the calling routine.

    Called by: RUART.
*/
```

```
GETBIT:   MOVEM.L D5-D6,-(SP)     ;
          MOVE.B  #1,D1           ;
          MOVE.L  $5AB5,D5        ;
          JMP     INTR_CHK+NEXT3  ;CHECK RS232 INPUT.
NEXT3:    NOP                     ;
          NOP                     ;PROVIDE ENOUGH DELAY FOR RS232 INTERRUPT.
          NOP                     ;
          JMP     ROM+EXIT3       ;
          NOP                     ;
EXIT3:    MOVE.L  $5AB6,D5        ;WITH THIS MOVE.L TO D5, THE FOLLOWING DBF
          MOVE.L  #1,D6           ; INSTRUCTIONS WITH 2 DIFFERENT COUNTS IN
          BTST    #0,D1           ; D6 GUARANTEES THE SAME AMOUNT OF DELAY
          BEQ.S   SKIP0           ; IN CASE OF AN INTERRUPT OCCURRENCE OR NOT.
          MOVE.L  #5,D6           ;
SKIP0:    DBF     D6,SKIP0        ;
          MOVE.L  #2,D6           ;
ADJ:      DBF     D6,ADJ          ;
          MOVE.L  $5AB7,D5        ;
          MOVEM.L (SP)+,D5-D6     ;
          RTS

/*  GETBIT Routine  Ends  */

/*  Communication Routines (SUART, RUART, DELAY, GETBIT) Ends  */


/*  FLTCLR Routine Below  */

    Function:  FLTCLR initializes the Coprocessor's control and data
               registers to zero. After power-up, the Coprocessor's
               registers contain ( $7FFF 000F FFFF FFFF ) in Packed Format.
               FLTCLR clears all Floating Point registers.
    Modified
    Registers: None.

    Called by: DOWNLOAD.
*/
FLTCLR:   WORD    $F23C           ;FMOVE.L #0,FP0
          WORD    $4000           ;
          WORD    $0000           ;
          WORD    $0000           ;
          WORD    $F23C           ;FMOVE.L #0,FP1
          WORD    $4080           ;
          WORD    $0000           ;
          WORD    $0000           ;
          WORD    $F23C           ;FMOVE.L #0,FP2
          WORD    $4100           ;
          WORD    $0000           ;
          WORD    $0000           ;
          WORD    $F23C           ;FMOVE.L #0,FP3
```

```
            WORD    $4180           ;
            WORD    $0000           ;
            WORD    $0000           ;
            WORD    $F23C           ;FMOVE.L #0,FP4
            WORD    $4200           ;
            WORD    $0000           ;
            WORD    $0000           ;
            WORD    $F23C           ;FMOVE.L #0,FP5
            WORD    $4280           ;
            WORD    $0000           ;
            WORD    $0000           ;
            WORD    $F23C           ;FMOVE.L #0,FP6
            WORD    $4300           ;
            WORD    $0000           ;
            WORD    $0000           ;
            WORD    $F23C           ;FMOVE.L #0,FP7
            WORD    $4380           ;
            WORD    $0000           ;
            WORD    $0000           ;
            RTS                     ;

/*  FLTCLR Routine Ends  */

/*  DOWNLOAD Routine Below  */

    Function:  DOWNLOAD, Downloads the bytes (in user program),
               which are sent by the Macintosh.
    Modified
    Registers: None.

    Called by: MAIN.
*/
DOWNLOAD: MOVE.B  #0,COP_ENB      ;
          BTST.B  #3,D3           ;COPROCESSOR ENABLED BY USER ?
          SNE     COP_ENB         ; ..YES, SET COPROCESSOR ENABLED FLAG.
          MOVEM.L D0-D7/A0-A7,-(SP);
          JSR     GETLONG         ;D3 DOWNLOAD ADDRESS.
          MOVEA.L D3,A1           ;[A1]<- LOAD ADDRESS
          MOVE.L  #8,D6           ;COUNTER TO SHIFT LOW BYTE TO HIGH BYTE
          JSR     RUART+ROM       ;GET HIGH BYTE OF COUNT.
          LSL.L   D6,D3           ;MOVE LOW BYTE TO HIGH.
          JSR     RUART+ROM       ;GET LOW BYTE OF COUNT.
          AND.L   #$0000FFFF,D3   ;CLEAR HIGH WORD.
          MOVE.L D3,D0            ;[D0]<- COUNT.
                                  ;THE NUMBER OF BYTES TO BE DOWNLOADED ARI
                                  ;EQUAL TO COUNT IN D0 ABOVE.
                                  ;NOW START TO RECEIVE COUNT NUM. OF BYTE:
DLOAD:    MOVE.L  A1,A3           ;SAVE LOAD ADDRESS FOR CHECKSUM CALCULA'
          MOVE.L  D0,D4           ;SAVE COUNT FOR CHECKSUM CALCULATION.
LAB99:    JSR     RUART+ROM       ;LOAD THE USER PROGRAM.
```

```
          MOVE.B   D3,(A1)+          ;
          SUB.W    #1,D0             ;
          BNE.S    LAB99             ;
          JSR      RUART+ROM         ;GET CHECKSUM.[D3.B]<- CHECKSUM.
          MOVE.B   D3,D5             ;SAVE RECEIVED CHECKSUM IN [D5.B].
          MOVE.L   A3,A1             ;RETRIEVE LOAD ADDRESS.
          MOVE.B   (A1)+,D0          ;BEGIN TO CALCULATE CHECKSUM.
          SUB.W    #1,D4             ;
LAB88:    MOVE.B   (A1)+,D6          ;
          EOR.B    D6,D0             ;
          SUB.W    #1,D4             ;
          BNE.S    LAB88             ;CHECKSUM CALCULATION DONE HERE.
          CMP.B    D5,D0             ;COMPARE RECEIVED & CALCULATED CHECKSUMS.
          BNE.S    C_ERROR           ;IN CASE OF ERROR ALERT MACINTOSH.
          BSR      FLTCLR            ;
          BSR      UPDATETBL2        ;LOAD REGISTER TABLE WITH CURRENT VALUES.
          BSR      UPLOAD            ;SEND THESE VALUES TO MACINTOSH.
          TST.B    COP_ENB           ;
          BEQ.S    SKP_C0            ;
          BSR      UPDTFLTBL         ;
          BSR      FUPLOAD           ;
SKP_C0:   MOVEM.L  (SP)+,D0-D7/A0-A7;
          RTS                        ;RETURN TO CALLER.
C_ERROR:  JSR      SENDERROR+ROM     ;
          MOVEM.L  (SP)+,D0-D7/A0-A7;
          RTS                        ;

/*  DOWNLOAD Routine Ends  */


/*  LDREGTBL Routine Below  */

    Function:  LDREGTBL receives the portion of data which contains register
               information. Places this data in the register table, which
               starts at the address TBL_D0.
    Modified
    Registers: None.

    Called by: GO.
*/
LDREGTBL:
          MOVEM.L  D0-D7/A0-A7,-(SP);
          MOVE.L   68(SP),D4         ;24 REGs x 4 = 96 BYTES WILL BE RECEIVED.
          MOVE.L   72(SP),A6         ;PUT BASE LOADING ADDRESS IN A6.
          MOVE.L   A6,A5             ;SAVE A6 IN A5 (FOR USE IN CHECKSUM CALC).
MORE:     JSR      RUART+ROM         ;START RECEIVING BYTES...
          MOVE.B   D3,(A6)+          ;
          SUB.L    #1,D4             ;
          BNE.S    MORE              ;RECEIVE ENDS...
          JSR      RUART+ROM         ;GET CHECKSUM BYTE.(IT WILL BE IN [D3]).
```

```
                                    ;COMPUTE CHECKSUM...
          MOVE.L   68(SP),D4        ;24 REGs x 4 = 96 BYTES WILL BE RECEIVED.
          MOVE.B   (A5)+,D0         ;              108 (FOR FLOATING)
          SUB.L    #1,D4            ;
GETCHKSUM:MOVE.B  (A5)+,D6          ;
          EOR.B    D6,D0            ;
          SUB.L    #1,D4            ;
          BNE.S    GETCHKSUM        ;
          CMP.B    D3,D0            ;CHECKSUM COMPUTATION ENDS.
          BNE.S    C_ERROR2         ;
          BRA      FINISH           ;RETURN TO CALLER.
C_ERROR2: JSR      SENDERROR+ROM    ;IN CASE OF ERROR ALERT MACINTOSH.
FINISH:   MOVEM.L (SP)+,D0-D7/A0-A7;
          RTS                       ;

/*  LDREGTBL Routine Ends  */


/*  UPLOAD Routine Below  */

    Function:  ULPOAD sends the data which is contained in register table,
               to the Macintosh.
    Modified
    Registers: None.

    Called by: DOWNLOAD, TRAPH, TRACEHANDLER.
*/
UPLOAD:   MOVEM.L D0-D7/A0-A7,-(SP);
          MOVEA.L #TBL_D0,A6        ;
          MOVE.B   #96,D4           ;SET BYTES COUNT IN D4.
ROUND:    MOVE.B   (A6)+,D3         ;
          JSR      ROM+SUART        ;
          MOVE.B   $6664,D3         ;THIS MOVE.B IS FOR ADJUSTING THE TIMING.
          SUB.B    #1,D4            ;[D0]<- COUNT.
          BNE.S    ROUND            ;
          MOVEM.L (SP)+,D0-D7/A0-A7;
          RTS                       ;RETURN TO CALLER.

/*  UPLOAD Routine Ends  */


/*  FUPLOAD Routine Below  */

    Function:  FULPOAD sends the data which is contained in Floating Point
               Register Table, to the Macintosh.
    Modified
    Registers: None.

    Called by: DOWNLOAD, TRAPH, TRACEHANDLER.
*/
```

```
FUPLOAD:  MOVEM.L  D0-D7/A0-A7,-(SP);
          MOVEA.L #TBL_FPCR,A6     ;
          MOVE.B  #108,D4          ;SET BYTES COUNT IN D4.
FROUND:   MOVE.B  (A6)+,D3         ;
          JSR     ROM+SUART        ;
          MOVE.B  $6666,D3         ;THIS MOVE.B IS FOR ADJUSTING THE TIMING.
          SUB.B   #1,D4            ;[D0]<- COUNT.
          BNE.S   FROUND           ;
          MOVEM.L (SP)+,D0-D7/A0-A7;
          RTS                      ;RETURN TO CALLER.

/*  FUPLOAD Routine Ends  */


/*  GO Routine Below  */

    Function:  GO receives the following parameters in that order:
               1- Display Steps Byte
               2- Five Break Point Addresses
               3- Five Break Point Counts
               4- All of the Registers, Program Counter.
    Modified
    Registers: None.

    Called by: MAIN.
*/
GO:       MOVE.B  #$AA,VIOL_FLAG  ;
          MOVE.B  D3,SAVECODE     ;
          MOVE.B  #1,FIRSTINST    ;
          JSR     RUART+ROM       ;GET DISP_STEP BYTE.
          MOVE.B  D3,DISP_STEP    ;PUT IT IN ITS LOCATION.
          MOVE.B  #5,D4           ;
          LEA     BRKPT1,A6       ;LOAD A6 WITH THE ADDRESS OF FIRST BREAK_
GETPTS:   JSR     GETLONG         ;POINT ADDRESS HOLDER.
          MOVE.L  D3,(A6)+        ;
          SUB.B   #1,D4           ;
          BNE.S   GETPTS           ;BRKPT1 THRU BRKPT5 ARE LOADED WITH
          MOVE.L  #8,D6           ; ADDRESSES.
          MOVE.B  #5,D4           ;
          LEA     BRKCNT1,A6      ;LOAD A6 WITH THE ADDRESS OF FIRST BRKCNT.
GETCTS:   JSR     RUART+ROM       ;GET LOW BYTE OF COUNT.
          LSL.L   D6,D3           ;MOVE LOW BYTE TO HIGH.
          JSR     RUART+ROM       ;GET LOW BYTE OF COUNT.
          AND.L   #$0000FFFF,D3   ;CLEAR HIGH WORD.
          MOVE.L  D3,(A6)+        ;[(A6)]<- COUNT.
          SUB.B   #1,D4           ;
          BNE.S   GETCTS          ;BRKCNT1 THRU 5 ARE LOADED WITH THE COUNTS.
          PEA     TBL_D0          ;
          MOVE.L  #96,-(SP)       ;
          BSR     LDREGTBL        ;LOAD THE REGISTER TABLE FROM MACINTOSH.
```

```
            ADD.L   #8,SP           ;
            TST.B   COP_ENB         ;IF USER DOES NOT ENABLE COPROCESSOR..
            BEQ.S   SKP_C1          ;DO NOT WAIT FOR FLOATING REGISTERS.
            PEA     TBL_FPCR        ;
            MOVE.L  #108,-(SP)      ;
            BSR     LDREGTBL        ;LOAD THE FLOATING REG. TABLE FROM
            ADD.L   #8,SP           ; MACINTOSH.
SKP_C1:     BTST.B  #7,TBL_SR       ;IS TRACE ALL ?
            BNE     PASS_6          ;...YES , DO NOT INSERT TRAP #15.
            BTST.B  #6,TBL_SR       ;IS TRACE BRANCH ?
            BNE     PASS_6          ;...YES , DO NOT INSERT TRAP #15.
            CMPI.L  #0,BRKPT1       ;IN CASE OF NO BREAK_POINT FOR ANY BRKPT
            BEQ.S   PASS_1          ;'$0000' WILL BE SENT FROM MAC. IF IT IS
            MOVE.L  BRKPT1,A6       ;ZERO THEN SKIP SAVING AND CHECK OTHERS.
            CMPA.L  TBL_PC,A6       ;IS BREAKPOINT = PC ?
            BEQ.S   PASS_1          ;... YES DO NOT INSERT TRAP_15 CODE.
            MOVE.W  (A6),TMPPT1+2   ;IF BRKPT1 IS NOT EQUAL TO $0000 THIS
            MOVE.W  #TRAP_15,(A6)   ;MEANS TAHT A BRKPT WILL OCCUR. AND FIRST
PASS_1:     CMPI.L  #0,BRKPT2       ;PIECE OF CODE IS TAKEN OUT AND SAVED IN
            BEQ.S   PASS_2          ;TMPPTx, THEN TRAP_15 CODE IS INSERTED.
            MOVE.L  BRKPT2,A6       ;
            CMPA.L  TBL_PC,A6       ;IS BREAKPOINT = PC ?
            BEQ.S   PASS_2          ;... YES DO NOT INSERT TRAP_15 CODE.
            MOVE.W  (A6),TMPPT2+2   ;
            MOVE.W  #TRAP_15,(A6)   ;
PASS_2:     CMPI.L  #0,BRKPT3       ;
            BEQ.S   PASS_3          ;
            MOVE.L  BRKPT3,A6       ;
            CMPA.L  TBL_PC,A6       ;IS BREAKPOINT = PC ?
            BEQ.S   PASS_3          ;.. YES DO NOT INSERT TRAP_15 CODE.
            MOVE.W  (A6),TMPPT3+2   ;
            MOVE.W  #TRAP_15,(A6)   ;
PASS_3:     CMPI.L  #0,BRKPT4       ;
            BEQ.S   PASS_4          ;
            MOVE.L  BRKPT4,A6       ;
            CMPA.L  TBL_PC,A6       ;IS BREAKPOINT = PC ?
            BEQ.S   PASS_4          ;.. YES DO NOT INSERT TRAP_15 CODE.
            MOVE.W  (A6),TMPPT4+2   ;
            MOVE.W  #TRAP_15,(A6)   ;
PASS_4:     CMPI.L  #0,BRKPT5       ;
            BEQ.S   PASS_5          ;
            MOVE.L  BRKPT5,A6       ;
            CMPA.L  TBL_PC,A6       ;IS BREAKPOINT = PC ?
            BEQ.S   PASS_5          ;... YES DON'T INSERT TRAP_15 CODE.
            MOVE.W  (A6),TMPPT5+2   ;
            MOVE.W  #TRAP_15,(A6)   ;
PASS_5:     BSR     UPDATEREGS      ;UPDATE REGISTERS FROM THE REGISTER TABLE
            TST.B   COP_ENB         ;
            BEQ.S   SKP_C2          ;
            BSR     UPDTFLREGS      ;
```

```
SKP_C2:   MOVE.L  TBL_USP,A0      ;READY FOR USP.
          MOVE.L  A0,USP          ;LOAD USP FROM TABLE.
          MOVE.L  TBL_SSP,A0      ;READY FOR SSP.
          LONG    $4E7B8803       ;LOAD SSP FROM REG.TABLE.
          MOVE.L  TBL_ISP,A0      ;READY FOR ISP.
          LONG    $4E7B8804       ;LOAD ISP FROM REG.TABLE.
                                  ;SKIP LOADING PC.
                                  ;DESIRED PC WILL BE LOADED WITH
                                  ;THE USE OF 'RTE' INSTRUCTION.
          CMP.B   #2,SAVECODE     ;
          BEQ.S   GOGO            ;
          BTST.B  #5,TBL_SR       ;
          BNE.S   N_USER          ;
          MOVE.L  TBL_USP,A0      ;
          MOVE.L  #CALL,-(A0)     ;
          MOVE.L  A0,USP          ;LOAD USP AFTER PUSHING CALL ADDRESS.
          BRA     GOGO            ;
N_USER:   BTST.B  #4,TBL_SR       ;
          BNE.S   N_INTR          ;
          MOVE.L  TBL_ISP,A0      ;
          MOVE.L  #CALL,-(A0)     ;
          LONG    $4E7B8804       ;LOAD ISP AFTER PUSHING CALL ADDRESS.
          BRA     GOGO            ;
N_INTR:   MOVE.L  TBL_SSP,A0      ;
          MOVE.L  #CALL,-(A0)     ;
          LONG    $4E7B8803       ;LOAD ISP AFTER PUSHING CALL ADDRESS.
GOGO:     MOVE.L  TBL_A0,A0       ;LOAD A0.IT WAS SKIPPED ABOVE.
          MOVE.W  #$0000,-(SP)    ;PUSH FORMAT/OFFSET WORD.
          MOVE.L  TBL_PC,-(SP)    ;PUSH THE PROGRAM COUNTER ON TO THE STACK.
          MOVE.W  TBL_SR,-(SP)    ;
          BSET.B  #7,(SP)         ;SET T1  (HIGH TRACE BIT).
          BCLR.B  #6,(SP)         ;CLEAR T0 (LOW TRACE BIT).
          RTE                     ;(T1 T0 = 1 0) ALLOWS TRACE ALL.
                                  ; THIS WILL POP THE NEW PC & SR VALUES
                                  ; OFF THE STACK AND CONTINUE EXECUTION
                                  ; WITH THE INSTRUCTION AT NEW PC VALUE.
PASS_6:   MOVE.B  #5,D4           ;
          LEA     BRKPT1,A6       ;LOAD A6 WITH THE ADDRESS OF FIRST BREAK
                                  ; POINT ADDRESS HOLDER.
CLR_ALL:  MOVE.L  #$00,(A6)+      ;
          SUB.B   #1,D4           ;
          BNE.S   CLR_ALL         ;
          BRA     PASS_5          ;

/*  GO Routine Ends  */

/*  MEMWRITE Routine Below  */

    Function:  MEMWRITE writes the user specified data (Byte/Word/LongWord)
```

```
                into user specified memory locations.
    Modified
    Registers: None.

    Called by: MAIN.
*/
MEMWRITE: MOVEM.L D0-D7/A0-A7,-(SP);
          JSR     RUART+ROM       ;GET SIZE.
          MOVE.B  D3,D4           ;[D4.B]<- SIZE.
          JSR     GETLONG         ;
          MOVEA.L D3,A6           ;[A6]<- MODIFY ADDRESS.
          MOVE.L  A6,A5           ;SAVE A6 IN A5.
          MOVE.B  D4,D5           ;
          BCLR    #7,D4           ;
CONTINUE: JSR     RUART+ROM       ;
          MOVE.B  D3,(A6)+        ;WRITE TO MEMORY, BYTE BY BYTE.
          SUB.B   #1,D4           ;LOOP UNTIL SIZE BYTES ARE RECEIVED.
          BNE.S   CONTINUE        ;(ie., FOUR BYTES IF SIZE = LONGWORD).
          BTST    #7,D5           ;BIT #7 OF SIZE BYTE IS SET BY MACINTOSH
          BNE.S   VERIFY          ;IF VERIFY WAS DESIRED.
          BRA     THERE           ;
VERIFY:   BCLR    #7,D5           ;IF USER WANTS TO VERIFY THEN READ BACK
TR_ALL:   MOVE.B  (A5)+,D3        ;AND SEND TO MACINTOSH.
          JSR     SUART+ROM       ;
          SUB.B   #1,D5           ;
          BNE.S   TR_ALL          ;
THERE:    MOVEM.L (SP)+,D0-D7/A0-A7;
          RTS                     ;RETURN TO CALLER.

/*  MEMWRITE Routine Ends  */


/*  MEMDISPLAY Routine Below  */

    Function:  MEMDISPLAY receives the user specified address range
               (16 bytes at a time), and fetches the bytes from these
               memory locations, and sends these bytes to Macintosh.
    Modified
    Registers: None.

    Called by: MAIN.
*/
MEMDISPLAY: MOVEM.L D0-D7/A0-A7,-(SP);
          JSR     GETLONG         ;
          MOVEA.L D3,A6           ;[A6]<- DISPLAY ADDRESS.
          JSR     RUART+ROM       ;GET LOW BYTE OF  COUNT.
          MOVE.B  D3,D4           ;[D4.B]<- COUNT.
                                  ;PARAMETERS ARE OBTAINED NOW, START TO SI
                                  ;THOSE MEMORY CONTENTS TO MACINTOSH.
          MOVE.B  (A6)+,D3        ;BYTE TO BE SENT IS IN D3 NOW.
```

```
          MOVE.B  D3,D0           ;
          JSR     SUART+ROM       ;SEND THE BYTE.
          SUB.B   #1,D4           ;
          BEQ.S   SKPTOUR         ;
TOUR_1:   MOVE.B  (A6)+,D3        ;BYTE TO BE SENT IS IN D3 NOW.
          EOR.B   D3,D0           ;CHKSUM WILL ACCUMULATE IN [D0.B].
          JSR     SUART+ROM       ;SEND THE BYTE, WHICH IS ALREADY IN D3.
          SUB.B   #1,D4           ;
          BNE.S   TOUR_1          ;
SKPTOUR:  MOVE.B  D0,D3           ;[D3] <-- CHKSUM ...
          JSR     SUART+ROM       ;SEND CHKSUM, WHICH IS ALREADY IN [D3.B].
          MOVEM.L (SP)+,D0-D7/A0-A7;
          RTS                     ;RETURN TO CALLER

/*  MEMDISPLAY Routine Ends  */


/*  UPDTFLTBL Routine Below  */

    Function:  UPDTFLTBL updates the Floating Point Register Table.
               It moves the control registers as longwords, and the
               Floating Point Registers as packed, to the table.
    Modified
    Registers: None.

    Called by: TRAPH, TRACEHANDLER.
*/
UPDTFLTBL:MOVE.L  A0,SAVEA0       ;SAVE A0 WITHOUT DISTURBING THE STACK.
          MOVE.L  #TBL_FPCR,A0    ;[A0]<- TABLE LOWER BASE ADDRESS.
WRT_FL:   LONG    $F218BC00       ;FMOVEM.L FPCR/FPSR/FPIAR,(A0)+
          LONG    $F2186C11       ;FMOVE.P  FP0,(A0)+
          LONG    $F2186C91       ;FMOVE.P  FP1,(A0)+
          LONG    $F2186D11       ;FMOVE.P  FP2,(A0)+
          LONG    $F2186D91       ;FMOVE.P  FP3,(A0)+
          LONG    $F2186E11       ;FMOVE.P  FP4,(A0)+
          LONG    $F2186EA1       ;FMOVE.P  FP5,(A0)+
          LONG    $F2186F21       ;FMOVE.P  FP6,(A0)+
          LONG    $F2186FA1       ;FMOVE.P  FP7,(A0)+
          MOVE.L  SAVEA0,A0       ;
          RTS                     ;

/*  UPDTFLTBL Routine Ends  */


/*  UPDATETBL Routine Below  */

    Function:  UPDATETBL updates the register table. Moves the copies
               of MC68020 registers, to the table.
    Modified
```

```
    Registers: None.

    Called by: TRAPH.
*/
UPDATETBL:MOVE.L  A0,SAVEA0        ;SAVE A0 WITHOUT DISTURBING THE STACK.
          MOVE.L  #TBL_USP,A0      ;[A0]<- TABLE LOWER BASE ADDRESS.
          MOVEM.L D0-D7/A0-A6,-(A0);LOAD ALL DATA REGS. A0 IS LOADED DUMMY
                                   ;IT WILL BE OVERWRITTEN ON NEXT LINE.
          MOVE.L  SAVEA0,TBL_A0    ;REAL VALUE OF A0 IS SAVED.
          MOVE.L  USP,A0           ;
          MOVE.L  A0,TBL_USP       ;USP IS LOADED.
          LONG    $4E7A8803        ;[A0]<- [MSP]
          MOVE.L  A0,TBL_SSP       ;LOAD MSP.
          LONG    $4E7A8804        ;[A0]<- [ISP]
          MOVE.L  A0,TBL_ISP       ;LOAD ISP.
          BTST.B  #4,TBL_SR        ;DID USER CHOOSE TO USE ISP ?..
          BNE.S   MSPTR            ;  OR MSP ?..
          ADD.L   #28,TBL_ISP      ;28 BYTES STACK SPACE IS USED BY:
          BRA     SKIPP            ;8 BYTES BY TRAP#15 4 WORD STACK FRAME,
MSPTR:    ADD.L   #28,TBL_SSP      ;16 BYTES BY SAVING D3, D4, A5, A6  REGS
                                   ;4 BYTES BY BSR UPDATETBL IN TRAP_ 15
                                   ; HANDLER.
SKIPP:    MOVE.L  22(SP),TBL_PC    ;LOAD PC.(IT IS AT LOCATION  SP+22).
          MOVE.W  #$00,TBL_SRHI    ;LOAD '$0000' FOR SR HIGH WORD.
          MOVE.W  20(SP),TBL_SR    ;LOAD SR LOW WORD. (IT IS AT LOCATION
                                   ; SP+20).
          LONG    $4E7A8801        ;
          MOVE.L  A0,TBL_VBR       ;LOAD VBR.
          LONG    $4E7A8002        ;
          MOVE.L  A0,TBL_CACR      ;LOAD CACR.
          LONG    $4E7A8802        ;
          MOVE.L  A0,TBL_CAAR      ;LOAD CAAR.
          MOVE.L  D0,SAVED0        ;
          LONG    $4E7A0001        ;[D0]<- DFC.
          MOVE.L  D0,TBL_SFC       ;DFC IS IN ITS PLACE.
          LONG    $4E7A0000        ;[D0]<- SFC.
          LSL.L   #4,D0            ;
          OR.L    D0,TBL_SFC       ;
          MOVE.L  SAVED0,D0        ;
          MOVE.L  SAVEA0,A0        ;
          RTS                      ;RETURN TO CALLER.

/*  UPDATETBL Routine Ends  */


/*  UPDATETBL2 Routine Below  */

    Function:  UPDATETBL2 updates the register table. Moves the copies
               of MC68020 registers, to the table. Slightly different from
               UPDATETBL.
```

```
    Modified
    Registers: None.

    Called by: DOWNLOAD.
*/
UPDATETBL2: MOVE.L  A0,SAVEA0      ;SAVE A0 WITHOUT DISTURBING THE STACK.
          MOVE.L  #TBL_USP,A0      ;[A0]<- TABLE LOWER BASE ADDRESS.
          MOVEM.L D0-D7/A0-A6,-(A0);LOAD ALL DATA REGS. A0 IS LOADED DUMMY
                                   ;IT WILL BE OVERWRITTEN ON NEXT LINE.
          MOVE.L  SAVEA0,TBL_A0    ;REAL VALUE OF A0 IS SAVED.
          MOVE.L  USP,A0           ;
          MOVE.L  A0,TBL_USP       ;USP IS LOADED.
          LONG    $4E7A8803        ;[A0]<- [MSP].
          MOVE.L  A0,TBL_SSP       ;LOAD MSP.
          LONG    $4E7A8804        ;[A0]<- [ISP].
          MOVE.L  A0,TBL_ISP       ;LOAD ISP.
          BTST.B  #4,TBL_SR        ;DID USER CHOOSE TO USE ISP ?..
          BNE.S   MSPTR2           ;  OR MSP ?..
          ADD.L   #$48,TBL_ISP     ;
          BRA     SKIPP2           ;
MSPTR2:   ADD.L   #$48,TBL_SSP     ;
SKIPP2:   MOVE.L  #$1000,TBL_PC    ;LOAD '$1000' FOR PC !...
          MOVE.W  #$00,TBL_SRHI    ;LOAD '$0000' FOR SR HIGH WORD.
          MOVE.L  #TBL_SR,A0       ;
          MOVE.W  SR,(A0)          ;LOAD SR LOW WORD.
          LONG    $4E7A8801        ;
          MOVE.L  A0,TBL_VBR       ;LOAD VBR.
          LONG    $4E7A8002        ;
          MOVE.L  A0,TBL_CACR      ;LOAD CACR.
          LONG    $4E7A8802        ;
          MOVE.L  A0,TBL_CAAR      ;LOAD CAAR.
          MOVE.L  D0,SAVED0        ;
          LONG    $4E7A0001        ;[D0]<- DFC.
          MOVE.L  D0,TBL_SFC       ;DFC IS IN ITS PLACE.
          LONG    $4E7A0000        ;[D0]<- SFC.
          LSL.L   #4,D0            ;
          OR.L    D0,TBL_SFC       ;
          MOVE.L  SAVED0,D0        ;
          MOVE.L  SAVEA0,A0        ;
          RTS                      ;RETURN TO CALLER.

/*  UPDATETBL2 Routine Ends  */


/*  UPDATETBL3 Routine Below  */

    Function:  UPDATETBL3 updates the register table. Moves the copies
               of MC68020 registers, to the table. Slightly different from
               UPDATETBL and UPDATETBL2.
    Modified
```

```
    Registers: None.

    Called by: TRACEHANDLER.
*/
UPDATETBL3: MOVE.L  A0,SAVEA0      ;SAVE A0 WITHOUT DISTURBING THE STACK.
          MOVE.L  #TBL_USP,A0      ;[A0]<- TABLE LOWER BASE ADDRESS.
          MOVEM.L D0-D7/A0-A6,-(A0);LOAD ALL DATA REGS. A0 IS LOADED DUMMY
                                   ;IT WILL BE OVERWRITTEN ON NEXT LINE.
          MOVE.L  SAVEA0,TBL_A0    ;REAL VALUE OF A0 IS SAVED.
          MOVE.L  USP,A0           ;
          MOVE.L  A0,TBL_USP       ;USP IS LOADED.
          LONG    $4E7A8803        ;[A0]<- [MSP].
          MOVE.L  A0,TBL_SSP       ;LOAD MSP.
          LONG    $4E7A8804        ;[A0]<- [ISP].
          MOVE.L  A0,TBL_ISP       ;LOAD ISP.
          BTST.B  #4,TBL_SR        ;DID USER CHOOSE TO USE ISP ?..
          BNE.S   MSPTR3           ;  OR MSP ?..
          ADD.L   #16,TBL_ISP      ;
          BRA     SKIPP3           ;
MSPTR3:   ADD.L   #16,TBL_SSP      ;xx BYTES STACK SPACE IS USED BY:
                                   ;8 BYTES BY TRAP#14 4 WORD STACK FRAME,
                                   ;16 BYTES BY SAVING D3, D4, A5, A6  REGS
                                   ;4 BYTES BY BSR UPDATETBL IN TRACE HANDLE
SKIPP3:   MOVE.L  6(SP),TBL_PC     ;LOAD PC. (IT IS AT LOCATION SP+6).
          MOVE.W  #$00,TBL_SRHI    ;LOAD '$0000' FOR SR HIGH WORD.
          MOVE.W  4(SP),TBL_SR     ;LOAD SR LOW WORD. (IT IS AT LOCATION SP
          LONG    $4E7A8801        ;
          MOVE.L  A0,TBL_VBR       ;LOAD VBR.
          LONG    $4E7A8002        ;
          MOVE.L  A0,TBL_CACR      ;LOAD CACR.
          LONG    $4E7A8802        ;
          MOVE.L  A0,TBL_CAAR      ;LOAD CAAR.
          MOVE.L  D0,SAVED0        ;
          LONG    $4E7A0001        ;[D0]<- DFC.
          MOVE.L  D0,TBL_SFC       ;DFC IS IN ITS PLACE.
          LONG    $4E7A0000        ;[D0]<- SFC.
          LSL.L   #4,D0            ;
          OR.L    D0,TBL_SFC       ;
          MOVE.L  SAVED0,D0        ;
          MOVE.L  SAVEA0,A0        ;
          RTS                      ;RETURN TO CALLER.

/*  UPDATETBL3 Routine Ends  */


/*  UPDTFLREGS Routine Below  */

    Function:  UPDTFLREGS updates the Floating Registers, with the data
               sent by the Macintosh.
```

```
    Modified
    Registers: None.

    Called by: GO.
*/
UPDTFLREGS:MOVE.L #TBL_FPCR,A0     ;
           LONG    $F2189C00       ;FMOVEM.L      (A0)+,FPCR/FPSR/FPIAR
           LONG    $F2184C00       ;FMOVE.P       (A0)+,FP0
           LONG    $F2184C80       ;FMOVE.P       (A0)+,FP1
           LONG    $F2184D00       ;FMOVE.P       (A0)+,FP2
           LONG    $F2184D80       ;FMOVE.P       (A0)+,FP3
           LONG    $F2184E00       ;FMOVE.P       (A0)+,FP4
           LONG    $F2184E80       ;FMOVE.P       (A0)+,FP5
           LONG    $F2184F00       ;FMOVE.P       (A0)+,FP6
           LONG    $F2184F80       ;FMOVE.P       (A0)+,FP7
           RTS                     ;

/*  UPDTFLREGS Routine Ends  */


/*  UPDATEREG  Routine Below  */

    Function:  UPDATEREGS updates the registers, with the data
               sent by the Macintosh.
    Modified
    Registers: None.

    Called by: GO.
*/
UPDATEREGS: MOVE.L  #TBL_D0,A0     ;
           MOVEM.L (A0)+,D0-D7     ;
           ADD.L   #4,A0           ;SKIP A0 IN THE TABLE (RESTORED IN "GO").
           MOVEM.L (A0)+,A1-A6     ;
           MOVE.L  TBL_VBR,A0      ;READY FOR VBR.
           LONG    $4E7B8801       ;LOAD VBR FROM REGISTER TABLE.
           MOVE.L  TBL_CACR,A0     ;READY FOR CACR.
           LONG    $4E7B8002       ;LOAD CACR FROM REGISTER TABLE.
           MOVE.L  TBL_CAAR,A0     ;READY FOR CAAR.
           LONG    $4E7B8802       ;LOAD CAAR FROM REGISTER TABLE.
           MOVE.L  TBL_SFC,D0      ;
           AND.L   #$0000000F,D0   ;
           LONG    $4E7B0001       ;[DFC]<-[D0].
           MOVE.L  TBL_SFC,D0      ;
           LSR.L   #4,D0           ;
           LONG    $4E7B0000       ;[SFC]<-[D0].
           MOVE.L  TBL_D0,D0       ;
           RTS                     ;SR WILL BE POPPED OFF THE STACK LATER.

/*  UPDATEREG  Routine Ends  */
```

```
/*  Interrupt Level 4 HANDLER  Routine Below  */

    Function:  HANDLER clears D1 to indicate a '0' has been received.
               A19, A17 bits of the return address are cleared to disable
               further interrupts, after RTE.
    Modified
    Registers: D1.

    Called by: In case of level 4 Interrupt.
*/
HANDLER:  ANDI.L  #$FFF5FFFF,2(SP);
          CLR.B   D1              ;
          RTE                     ;

/*  Interrupt Level 4 HANDLER  Routine Ends  */


/*  TRAP HANDLER Routine Below  */

    Function:  TRAPH, Handles Trap 15.
               Puts all Registers & Stack Pointer Contents to Memory,
               (namely to the register table), and waits for Command
               from Macintosh.
               TRAP Instruction
               SSP-2 -> SSP    FORMAT/VECTOR OFFSET -> (SSP)
               SSP-4 -> SSP                      PC -> (SSP)
               SSP-2 -> SSP                      SR -> (SSP)
                               VECTOR ADDRESS -> PC
    Modified
    Registers: SP, SR.

    Called by: In case of Trap 15 Occurs.
*/
TRAPH:    MOVEM.L D3-D4/A5-A6,-(SP);SAVE THE REGISTERSTO BE MODIFIED.
          SUB.L   #2,18(SP)        ;[PC]<- [PC]-2 (THE ONE SAVED ON STACK).
          BSR     UPDATETBL        ;LOAD REG TABLE BEFORE  UPLOADING IT.
          TST.B   COP_ENB          ;IF USER DOES NOT ENABLE COPROCESSOR
          BEQ.S   SKP_C3           ;DO NOT UPDATE FLOATING REGISTER TABLE.
          BSR     UPDTFLTBL        ;
SKP_C3:   MOVE.L  18(SP),D3        ;[D3]<- INSTRUCTION ADDR. CAUSING TRAP_1
          MOVE.W  #4,D4            ;
          LEA     BRKPT1,A6        ;ONE OF THE BREAKPOINTS SHOULD BE EQUAL
SEARCH:   CMP.L   (A6)+,D3         ;TO THAT ADDRESS.
          DBEQ    D4,SEARCH        ;
          CMP.W   #0,D4            ;
          BLT     DSPLY            ;
          SUB.L   #4,A6            ;IF SO DECREMENT TAHT BREAKPOINT'S COUNT
          MOVE.L  (A6),A5          ;
          MOVE.W  -18(A6),(A5)     ;PUT THE ORIGINAL CODE BACK TO ITS PLACE
```

```
          CMPI.L  #0,20(A6)          ;IF THE BREAKCOUNT IS ZERO, DON'T DECREMENT
          BEQ.S   NOT_SUB            ;IT. SO EVERY TIME THAT ADDRESS IS REACHED
          SUB.L   #1,20(A6)          ;A BREAKPOINT WILL OCCUR.
NOT_SUB:  CMP.B   #0,DISP_STEP       ;IF DISPLAY_STEP IS SET, THEN DISPLAY THE
          BNE.S   DSPLY              ;RESULTS TO THE USER EACH TIME THAT
          CMP.L   #0,20(A6)          ;IS REACHED, REGARDLESS OF ITS COUNT.
          BNE.S   SKPDSPLY           ;IF DISPLAY_STEP IS NOT SET, THEN DISPLAY
DSPLY:    BSR     UPLOAD             ;ONLY WHEN ITS COUNT DECREMENTS TO ZERO,
          BSR     SCNTS              ;SEND BACK THE MOST RECENT BREAKCOUNTS.
          TST.B   COP_ENB            ;
          BEQ.S   SKP_C4             ;
          BSR     FUPLOAD            ;
SKP_C4:   MOVE.L  #5,D4              ;
          LEA     BRKPT1,A6          ;
LOOK:     CMP.L   #0,(A6)+           ;RESTORE ALL ORIGINAL INSTRUCTION CODES
          BEQ.S   DO_NTH             ;HAVING BREAKPOINTS BEFORE RETURNING TO
          MOVE.L  -4(A6),A5          ;MAIN.
          MOVE.W  -22(A6),(A5)       ;
DO_NTH:   SUB.L   #1,D4              ;
          BNE.S   LOOK               ;
          MOVE.L  #MAIN,18(SP)       ;IF DISPLAYED THEN LOOP IN MAIN WAITING FOR
          BRA     RESTORE            ;THE NEXT COMMAND. (SO PUT MAIN ADDR. IN
                                     ;ITS PLACE ON THE STACK). OTHERWISE DON'T
                                     ;RETURN TO MAIN PGM, INSTEAD CONTIUNE WITH
                                     ;THE EXECUTION OF THE NEXT INSTRUCTION.
SKPDSPLY: BSET.B  #7,16(SP)          ; SET T1 OF STAUS REGISTER.
          BCLR.B  #6,16(SP)          ; CLEAR T0 OF STATUS REGISTER. (TRACE ALL).
RESTORE:  BCLR.B  #4,16(SP)          ;
          BSET.B  #5,16(SP)          ; WILL BE IN SUPERVISOR MODE ON EXIT.
          MOVEM.L (SP)+,D3-D4/A5-A6;
          RTE

/*  TRAP HANDLER Routine Ends  */


/*  Interrupt Level 6 (ABORT) HANDLER  Routine Below  */

    Function:  ABORT arranges the Stack (for compatibility with the TRAP
               HANDLER Routine), and branches to TRAPH.
    Modified
    Registers: SP, SR.

    Called by: In case of Level 6 Interrupt, which is generated to
               provide ABORT.
*/
ABORT:    ORI     #MASK_7,SR         ;DISABLE INTERRUPTS.
          ANDI.W  #$2FFF,SR          ;DISABLE TRACE.
          ADDI.L  #2,2(SP)           ;COMPENSATE FOR SUBTRACTION.
          BRA     TRAPH              ;CONTINUE WITH TRAPH.
```

```
/*  Interrupt Level 6 (ABORT) HANDLER  Routine Ends  */


/*  STACKFRAME  Routine Below  */

    Function:  STACKFRAME just arranges the stack. The address of this
               routine is placed in the exception vector table entries,
               for unimplemented exceptions. The purpose is to prevent
               system crash, when those unimplemented exceptions occur.
    Modified
    Registers: SP.

    Called by: In case of unimplemented exceptions.
*/
STACKFRAME:ANDI.W #$2FFF,(SP)      ;DISABLE TRACE.
           ANDI.W  #INTR_ENB,(SP)  ;ENABLE INTERRUPTS.
           MOVE.L  #MAIN,2(SP)     ;
           RTE                     ;

/*  STACKFRAME  Routine Ends  */


/*  GETLONG  Routine Below  */

    Function:  GETLONG receives a longword, which is sent by the Macintosh.
    Modified
    Registers: D3, which passes the received longword to the calling routir

    Called by: DOWNLOAD, GO, MEMWRITE, MEMDISPLAY.
*/
GETLONG:  MOVEM.L D0-D2/D4-D7/A0-A7,-(SP);
          MOVE.L  #8,D6            ;COUNTER TO SHIFT LOW BYTE TO HIGH BYTE.
          JSR     RUART+ROM        ;GET BYTE #3 OF LOAD ADDRESS.
          LSL.L   D6,D3            ;SHIFT IT TO ITS PLACE.
          JSR     RUART+ROM        ;GET BYTE #2 OF LOAD ADDRESS.
          LSL.L   D6,D3            ;
          JSR     RUART+ROM        ;GET BYTE #1 OF LOAD ADDRESS.
          LSL.L   D6,D3            ;MOVE BYTE #1 ITS POSITION.
          JSR     RUART+ROM        ;GET BYTE #0 OF LOAD ADDRESS. ( LS BYTE )
          MOVEM.L (SP)+,D0-D2/D4-D7/A0-A7;
          RTS                      ;RETURN TO CALLER.

/*  GETLONG  Routine Ends  */


/*  SENDERROR  Routine Below  */

    Function:  SENDERROR sends a lot of successive zeros, which will
               cause a Frame Error, and its detection on the Macintosh.
               So, Macintosh will know that something went wrong during
```

```
            transmission of data to the ECB.
    Modified
    Registers: D0,D1.

    Called by: RUART, DOWNLOAD, LDREGTBL.
*/
SENDERROR:MOVE.B  #BRKCOUNT,D0    ;THAT MANY TIMES ZERO BITS WILL BE SENT.
STEP1:    JSR     SEND_ZER+DELAY1 ;SEND A ZERO.
          SUB.B   #1,D0           ;
          BNE.S   STEP1           ;
          MOVE.W  #MAXINT,D0      ;
STEP4:    MOVE.W  #10,D1          ;FOR 10 x 100 MICRO SECOND DELAY.
STEP2:    MOVE.L  #7,D2           ;
STEP3:    DBF     D2,STEP3        ;100 MICRO SECOND DELAY.
          SUB.W   #1,D1           ;
          BNE.S   STEP2           ;
          SUB.W   #1,D0           ;
          BNE.S   STEP4           ;
          RTS                     ;

/*  SENDERROR  Routine Ends  */


/*  SCNTS  Routine Below  */

    Function:  SCNTS sends the most updated BreakCounts to the Macintosh.
    Modified
    Registers: None.

    Called by: TRACEHANDLER, TRAPHANDLER.
*/
SCNTS:    MOVEM.L D3-D4/A6,-(SP)  ;
          MOVE.L  #5,D4           ;
          MOVE.L  #BRKCNT1+2,A6   ;
FORALL:   MOVE.B  (A6)+,D3        ;
          JSR     ROM+SUART       ;
          MOVE.B  (A6)+,D3        ;
          JSR     ROM+SUART       ;
          ADDA.L  #2,A6           ;
          SUB.L   #1,D4           ;
          BNE.S   FORALL          ;
          MOVE.B  VIOL_FLAG,D3    ;SEND PRIVILAGE VIOLATION CODE.($55 FOR YES,
          JSR     ROM+SUART       ; $AA FOR NO).
          MOVEM.L (SP)+,D3-D4/A6  ;
          RTS

/*  SCNTS  Routine Ends  */
```

```
/*  TRACEHANDLER  Routine Below  */

    Function:  TRACEHANDLER handles the Trace case.
    Modified
    Registers: SP, SR.

    Called by: In case of Trace (Trace All or Trace Branch).
*/
TRACEHANDLER: MOVEM.L D3-D4/A5-A6,-(SP);
          MOVE.L  24(SP),D3       ;[D3] = FAULTING INSTRUCTION ADDRESS.
          MOVE.W  #4,D4           ;
          LEA     BRKPT1,A6       ;
SEEK:     CMP.L   (A6)+,D3        ;
          DBEQ    D4,SEEK         ;
          CMP.W   #0,D4           ;
          BLT     SRC_FAIL        ;
MATCH:    SUB.L   #4,A6           ;
          MOVE.L  (A6),A5         ;
          MOVE.W  (A5),-18(A6)    ;
          MOVE.W  #TRAP_15,(A5)   ;
SRC_FAIL: MOVEM.L (SP)+,D3-D4/A5-A6;
          BTST.B  #6,TBL_SR       ;
          BNE.S   TRC_BRA         ;
          BTST.B  #7,TBL_SR       ;
          BNE.S   TRC_ALL         ;
          BRA     NO_TRACE        ;
TRC_BRA:  CMP.B   #1,FIRSTINST    ;
          BEQ.S   NOT_SHOW        ;
          BSR     UPDATETBL3      ;
          BSR     UPLOAD          ;
          BSR     SCNTS           ;
          TST.B   COP_ENB         ;IF USER DOES NOT ENABLE COPROCESSOR
          BEQ.S   SKP_C5          ;
          BSR     UPDTFLTBL       ;
          BSR     FUPLOAD         ;
SKP_C5:   ANDI.W  #$2FFF,(SP)     ;DISABLE TRACE. T1-T0 -> NO TRACE.
          BSET.B  #5,(SP)         ;WILL BE IN SUPERVISOR MODE ON EXIT.
          MOVE.L  #MAIN,2(SP)     ;WILL RETURN TO MAIN PROGRAM.
          BRA     FINE            ;
NOT_SHOW: BCLR.B  #7,(SP)         ;SHOULD BE TRACE ALL.
          BSET.B  #6,(SP)         ;T1-T0 -> TRACE ALL.
          BRA     FINE            ;
TRC_ALL:  BSR     UPDATETBL3      ;
          BSR     UPLOAD          ;
          BSR     SCNTS           ;SEND THE MOST RECENT BREAKCOUNTS.
          TST.B   COP_ENB         ;IF USER DOES NOT ENABLE COPROCESSOR
          BEQ.S   SKP_C6          ;
          BSR     UPDTFLTBL       ;
          BSR     FUPLOAD         ;
SKP_C6:   ANDI.W  #$2FFF,(SP)     ;CLEAR TRACE BITS NOT TO TRACE OURSELVES
```

```
          BSET.B   #5,(SP)          ;WILL BE IN SUPERVISOR MODE ON EXIT.
          MOVE.L   #MAIN,2(SP)      ;WILL RETURN TO MAIN PROGRAM.
          BRA      FINE             ;
NO_TRACE: BCLR.B   #7,(SP)          ;SINCE USER WANTS NO TRACE, CLEAR T1.
FINE:     MOVE.B   #0,FIRSTINST     ;NOT FIRST INSTRUCTION ANYMORE.
          RTE                       ;

/*  TRACEHANDLER  Routine Ends  */


/*  VIOLHANDLER  Routine Below  */

    Function:  VIOLHANDLER handles Privilage Violations.
    Modified
    Registers: SP.

    Called by: In case of Privilage Violation.
*/
VIOLHANDLER:
          MOVE.B   #$55,VIOL_FLAG   ;
          ADD.L    #2,2(SP)         ;COMPENSATE FOR THE SUBTRACTION FOR
                                    ;BREAKPOINTS IN TRAPH ROUTINE.
          BRA      TRAPH            ;SINCE BOTH PRIVILEGE VIOLATION AND THE
                                    ;TRAP_15 HAVE THE SAME STACK FRAME.
                                    ;PC POINTS TO FAULTING INSTRUCTION.

/*  VIOLHANDLER  Routine Ends  */

/*  CALL (Subroutine Test)  Below  */

CALL:     WORD     TRAP_15          ;

/*  CALL (Subroutine Test)  Ends  */

/*  MEMORY ALLOCATION  */


TMPPT1:   LONG     $0000            ;THIS WILL BE USED FOR SAVING CODE PARTS
TMPPT2:   LONG     $0000            ;TAKEN OUT OF CODE FOR TRAP15 INSERTION.
TMPPT3:   LONG     $0000            ;TMPPT1 WILL HOLDTHE PIECE OF CODE TAKEN
TMPPT4:   LONG     $0000            ;OUT FOR INSERTION A TRAP_15 CODE FOR
TMPPT5:   LONG     $0000            ;BREAKPOINT #1 (BRKPT1).

BRKPT1:   LONG     $0000            ;THIS WILL BE USED FOR STORING THE
BRKPT2:   LONG     $0000            ;ADDRESSES AT WHICH THE BREAKPOINT
BRKPT3:   LONG     $0000            ;WILL OCCUR.
BRKPT4:   LONG     $0000            ;(CORRESPONDING TO 5 DIFFERENT BREAK_
BRKPT5:   LONG     $0000            ;POINTS).

BRKCNT1:  LONG     $0000            ;THE BREAKPOINT COUNTS ASSOCIATED
```

```
BRKCNT2:   LONG    $0000          ;WITH EACH BRAEKPOINT WILL BE STORED
BRKCNT3:   LONG    $0000          ;AT THESE BRKCNTx (1 THRU 5).
BRKCNT4:   LONG    $0000          ;
BRKCNT5:   LONG    $0000          ;

SAVEA0:    LONG    $0000          ;A0 WILL BE SAVED HERE TEMPORARILY.
SAVED0:    LONG    $0000          ;D0 WILL BE SAVED HERE TEMPORARILY.
SAVESR:    WORD    $0000          ;THE STATUS REG. WILL BE SAVED HERE.
SAVECODE:  BYTE    $00            ;TEMP STORAGE FOR MAC CODE.

FIRSTINST:BYTE     $00            ;THIS IS FOR FIRST INSTRUCTION WHICH
                                  ; WILL BE TRACED FOR SINGLE STEP.
DISP_STEP:BYTE     $00            ;WILL THE STEPS BE DISPLAYED OR NOT ?..
VIOL_FLAG:BYTE     $00            ;PRIVILAGE VIOLATION FLAG.
COP_ENB:   BYTE    $00            ;USER WANTS TO USE COPROCESSOR.
LAST:      NOP                    ;
           END

/*  ECB ROM Resident Routines End  */
```

APPENDIX D: SERIAL COMMUNICATION IN SOFTWARE

RECEIVING

Level four interrupt is used to sense the RS232 input. An interrupt is generated when a logic one is present at RS232 input. But before this happens, address lines A19 and A17 have to be made high, thus enabling the AND gate which produces the level four interrupt.

In order to enable this interrupt, first the address lines A17 and A19 are forced to be HIGH, which is done by JMP INTR_CHK+NEXTx, in RUART routine. But, since some amount of time is needed to acknowledge an interrupt, several NOP instructions are added following the JMP INTR_CHK+NEXTx instruction. This guarantees that previous address stay unchanged while the microprocessor executes these NOPs. By doing this, the address bits A17 and A19 are kept high enough for the interrupt to be acknowledged by the CPU.

How incoming bits are sensed ?

The time that CPU spends by executing the JMP INTR_CHK+NEXTx, and the following several NOP instructions can be considered as a sampling window. If an interrupt occurred during the sampling window, program execution is continued with the level four interrupt handler routine. This routine first forces the address lines A19 and A17 to zero, thus disabling the AND gate which senses the RS232 line. As a consequence, further interrupts are disabled.

Following this instruction, routine clears register D1. After RTE, instruction execution continues from where it was previously. Then register D1, which is set to one before receiving each incoming bit, is tested. If its content is zero, this shows that a level four interrupt did occur, which means a logical zero is received from RS232 input. Otherwise, if D1 still contains a one, this means that a logical one is received.

How incoming bytes are received ?

The receiving routine, RUART, looping all the time, checks for the RS232 input. RS232 line, when it is idle, stays at high voltage level. After sensing the start bit, eight bits are received and shifted in to lower byte of D3. The reception of that byte ends with the detection of the stop bits. If a frame error occurs during reception, RS232 input to the Macintosh is kept low for a while and the user is alerted.

More detailed information can be obtained from Appendix C. (Source code of ecb.asm).

TRANSMITTING

SUART routine sends a byte which is in D3. In SUART, first by the instruction JSR SEND_ZER+DELAY1, by sending a zero bit the start bit is sent. Here, DELAY1 subroutine provides 104.7 microseconds delay between the bits to be transmitted. Following the start bit, eight bits are sent which are the bits in the lower byte of D3. JSR SEND_ONE+DELAY1 or SEND_ZER+DELAY1 is used in order to send a ONE or a ZERO bit.

More detailed information can be obtained from Appendix C (Source code of ecb.asm).

APPENDIX E: IMPLEMENTATION OF SOFTWARE ABORT

ABORT

Abort is a very beneficial option to the user. About its implementation on the ECB: when the Abort button is pressed, a level six autovectored interrupt is generated.

During the first design phase of the debugger it was intended to support the Abort in software. If a long enough Break could be sent to the ECB, then this could be interpreted as a user intention for Abort. This idea did not work. Because when the user program enters in to an endless loop or just gets out of control (these two situations can occur right after Go menu), since the Macintosh will still be waiting for information from the ECB (which will never come), the Macintosh will be locked and there is no way to get out of Go menu and send a Break to the ECB. For this reason, the Abort option was decided to be implemented in hardware.

At this point, the design idea was to make this interrupt, a level seven, non-maskable interrupt. But later it was noticed that it would not work. Because pressing the Abort button once, caused many interrupts, each non-maskable. To overcome this problem, using a debouncing circuit could be a choice, but the tradeoff was more hardware. For this reason a level six interrupt was found to be appropriate. In this way, when the Abort button is pressed once, it still creates many level six interrupts, but only the first one is processed and the rest is ignored. At the entry on the Abort handler routine, interrupts are disabled, so the interrupts caused by the bouncing of the Abort button is ignored.

APPENDIX F: OPERATING INSTRUCTIONS

INSTRUCTIONS

1- With Macintosh off, insert Disk "C Compiler" into Floppy Disk Slot.
2- Turn on Macintosh. Macsbug is installed, Lightspeed C is started immediately.
3- Now, you may select the project. Double-Click the project file: Tutor20 pi. (Double-Click Tutor20 pi means; move the mouse so the arrow for the mouse is on the line with the name Tutor20 pi, and click the button on top of the mouse, twice, quickly.)
On the upper right corner of the screen, you can see the files, contain in this project.
4- In order to be able to create your assembly language program, double-click on the file "test.c".
5- Now, you may start editing your program. But, it is advisable to make test.c the only file you work on, saving other files, if they exist, using other file names. To save a copy of test.c as backup.c and then be able to modify test.c, drag File to Save A Copy As (Drag File to Save A Copy As... means, using the mouse, click button down on the "File" menu, hold the button down as mouse is moved down to "Save A Copy As..." and release the mouse.)
Type the name of the new file (Actually, a copy of test.c), backup.c and hit carriage return.
6- Do not alter any of the lines, unless told otherwise. User program area is clearly shown in test.c. User should type his program between the lines "*** USER PROGRAM ***" and "*** USER PROGRAM ENDS ***".
7- Labels begin with a "@" sign, followed by digits. (You are not allowed t use labels @1, @2, @3, which are already defined and used by test.c program.)
Hexadecimal numbers begin with a "0x" (Zero Eks), followed by digits or a-f or A-F.
Variables (e.g., i) are declared in C above the line "asm{", (e.g., int and are accessed using index addressing with address register A6 and a negative offset.
8- Drag project to Run. Test.c will be compiled and the debugger will run. You should see Apple, File, Functions menus on the screen.
9- If you want to utilize MC68881 Coprocessor, or if you want to have a hard copy of what is going to be displayed on the screen, or some other options; pull down Functions menu and click the mouse on Options menu. Here, you will see a button corresponding to each option. You can have any option "ON" by just clicking it (When it is darkened, that means you have that option, or vice versa). Click "Quit" to get out of that menu. If you want to use Coprocessor instructions in your program, you have to

have the "Coprocessor" option at this step, before Downloading. You are not allowed to first Download and then select "Coprocessor" option. This will lock the system.

10-Pull down Functions menu and click the mouse on Download option. This will download your program to the ECB. Your program will be loaded in RAM starting at 1000 (Hexadecimal) address.

11-Now you need to select "Go" menu. The default Program Counter value is 1000 Hex. You may change this address if you want to.
A very important point needs to be explained. That is, if you are going to select "Goto" option within the "Go" menu, your program has to end with a TRAP #15 instruction, or if you're going to select "Call" option an RTS instruction should be at the end of your program.

WHAT CAN YOU DO IN A PARTICULAR MENU ?

When you pull down the Functions menu, you will see the following selections.

01-Download
02-Go
03-Registers
04-Floating Regs
05-Memory Display
06-Memory Write
07-Options
08-Previous Screen
09-Clear Screen
10-Help

DOWNLOAD

Downloads the user program from the Macintosh to the ECB, at a Baud rate of 9600. After that, the current register values, whatever they were, are uploaded from the ECB.
If Coprocessor will be used, that option should be selected in the Options menu, before clicking Download.

GO

A- GoTo/Call:
There are two types of program execution, as far as the procedure is concerned. They are:
1- Goto
2- Call

The user as to end his program either with RTS or with TRAP #15, depend on the situation. This was described above, at step 11. The purpose of "Call" is that, with this choice, the user can easily test and run his subroutines.

B- Return to:
User has choices about which menu to go after that part of execution of his program. the default return menu is "No Menu" where no menu displayed, instead the register values, and the instruction following the last executed instruction in a disassembled form are displayed.
Clicking on "Return to", "Registers Menu" is selected, which simply displays the "Register Menu".
Clicking on "Return to" a second time, "Go Menu" is selected, which makes the same menu appear again.

C- Breakpoints:
User can set, upto five breakpoints. The "Clear All" option, clears all the breakpoints. Hitting the tab, the darkened spot passes through the breakcounts first, where user can enter the count he or she wants. This number can be in the range (0..9999). After Breakcounts, Breakpoi addresses can be entered, just by typing the desired address and hitti the tab. If no breakcount was entered for this breakpoint before, its value is set to one, automatically.

D- Display Steps:
If this option is made "ON", just by clicking it, every step taken during the program execution on the ECB is displayed on the Macintosh. This situation may be useful when the user sets any Breakcount to a valu bigger than one, and still wants to see the outcome of every single ste If this option were not used, the information would be displayed after t Breakpoint address is reached as many as Breakcount times.

E- Cancel:
Anything done during this Go Menu session is ignored.

F- Go:
A final step in Go Menu. Clicking "Go" will download the most updated register values, breakpoint information, and then the program execution will start.

REGISTERS

When selected displays MC68020 register information, interrupt leve and condition codes.

A- Registers:
All the data, address and control registers are displayed. Any of these registers can be modified just by entering the desired content and hitting the tab.

B- Clear All:
When clicked, clears all data registers, and all address registers except A7, which is the stack pointer.

C- Active Stack Pointer:
A7 entry shows the active stack pointer. Default is "User Stack Pointer'

Clicking A7 once, switches to "Interrupt Stack Pointer". Clicking A7 once again, switches to "Supervisor Stack Pointer".

D- Condition Codes:
Displayed as radio buttons, where darkened one means that bit is set. User can change condition code values either by clicking it, or by modifying the Status Register.

E- Go:
When clicked, does the same function as what it would do in "Go" menu. But there is a condition. The Registers menu should have been called by "Go" menu, which means that, in Go menu "Return to" field was "Registers menu" before the last clicking of Go. This option is provided just for presenting some ease to user. Because this way he does not need to go through "Go menu". If the above condition does not hold, clicking "Go" does not mean anything, nothing happens.

F- Interrupt Level:
Every clicking, increases the interrupt level by one. This field can also be changed by modifying Status Register. Since it will crash the system, the user is not allowed to set the Interrupt Level to a value greater than three. (Level four interrupt is used for establishing serial communication. If a higher level interrupt were allowed this could crash serial communication mechanism).

G- Quit:
Simply quits the Registers menu.

FLOATING REGS

When selected, MC68881 Floating Point Register information, FPCR (Floating Point Control Register), FPSR (Floating Point Status Register), FPIAR (Floating Point Instruction Address Register), condition codes, Exception Status/Enable byte are displayed. This menu can be selected only when Coprocessor option is turned On in the Options menu.
Eight Floating Point Registers are displayed, each consisting of four fields. These fields are: Exponent, sign of exponent, mantissa, sign of mantissa. Each field is modified by the user separately. User may modify FPCR, FPSR, FPIAR, by typing the desired value and hitting the tab. User may also modify condition codes, or Exception Status/Enable bits by just clicking them.

A- Quit:
Simply quits the Floating Regs menu.

MEMORY DISPLAY

The maximum number of bytes to be displayed at once is 500.

A- From:
The beginning address of memory display needs to be entered here, by just typing that address and hitting the tab.

B- To:
The ending address of memory display needs to be entered here, by just typing that address and hitting the tab.

C- Size:
Size is the number of bytes to displayed, which is automatically calculated and displayed (size=from-to). Entering any one of from or "to", and "size" will work as well.

D- Disassemble:
The memory is displayed in a disassembled format, one instruction per line.

E- Cancel:
Simply ignores that Memory Display session.

F- Display:
Clicking Display, causes the display of the desired memory locations.

MEMORY WRITE

When selected, user is able to modify any memory location. That memory location can be Byte, Word (two bytes), or Longword (four bytes) in length. In case Increment/Decrement option is selected, "Location" is Incremented/Decremented by one, two, or four, according to the data length being modified.
When "No change" is selected, "Location" is not modified, so following writes occur to the same memory location.

A- Location:
The address of the memory location to be modified. User can enter the address by just typing it and then hitting the tab.

B- Contents:
Here, user has to type the new content of that memory location. The memory write is done only, when the user hits the tab.

C- Verify:
   When selected, a memory write to that location is done, following that, a memory read is performed from the same memory location. This value is sent back to the Macintosh, where it is compared against the desired content, by the debugger. If an error is detected, user is alerted.

D- Quit:
   Simply quits the Memory Write menu.

OPTIONS

When this menu is selected four options will be displayed. Clicking any of these options, will toggle it (ie., turning it ON and OFF). The following describes what is done when any particular option is selected.

A- Hardcopy:
   Whatever seen on the screen is also sent to a serial printer. This option might be useful especially when user dumps large number of bytes of memory.

B- Coprocessor:
   If user wants to access Coprocessor this option should be selected before Downloading. Turning this option on enables the "Floating Regs" menu, which would not be accessible by the user, otherwise.

C- Refresh Screen:
   Following a Quit from any menu, instead of displaying a blank screen, a screenful information is displayed. This information is obtained from a circular queue, which contains the last displays on the screen.

D- Experienced:
   If the user is not experienced he is not allowed to change interrupt levels.

PREVIOUS SCREEN

When selected, the last screenful information is sent to the serial printer. In a sense, it is like a hardcopy of Refresh Screen.

CLEAR SCREEN

When selected, clears the screen.

HELP

Displays help information.

APPENDIX G: SAMPLE ASSEMBLY LANGUAGE PROGRAMS

A- Sample Program #1

The following program, copies the elements of ARRAY_A to ARRAY_B. Each element is one byte long.

i- Source Code

```
;     ** Sample Program #1 **

;Label   Opcode    Operand          Comment
;Field   Field     Field            Field

ARRAYA: BLKB      5                ;DEFINE ARRAY_A
ARRAYB: BLKB      5                ;DEFINE ARRAY_B
;
;ASSUME ARRAY_A HAS SOME VALUES
;
        LEA       ARRAYA,A0        ;A0 POINTS TO ARRAY_A
        LEA       ARRAYB,A1        ;A1 POINTS TO ARRAY_B
        MOVE.B    #5,D0            ;FIVE ELEMENTS TO BE COPIED
LOOP:   MOVE.B    (A0)+,(A1)+      ;COPY ELEMENT OF A TO ELEMENT OF B
        SUB.B     #1,D0            ;DECREMENT THE COUNTER
        CMP.B     #0,D0            ;FIVE OF THEM COPIED ?..
        BNE.S     LOOP             ;NO... COPY ONE MORE.
DONE:
```

ii- Listing

```
            2500 A.D. 68000 Macro Assembler - Version 4.03a
            -----------------------------------------------

                    Input  Filename : sample1.asm
                    Output Filename : sample1.obj


;     ** Sample Program #1 **
                          ;Label  Opcode   Operand  Comment
                          ;Field  Field    Field    Field


00000000                  ARRAYA: BLKB     5          ;DEFINE ARRAY_A
00000005                  ARRAYB: BLKB     5          ;DEFINE ARRAY_B
;
;ASSUME ARRAY_A HAS SOME VALUES
;
0000000A  41F9 0000 0000          LEA    ARRAYA,A0   ;A0 POINTS TO ARRAY_A
00000010  43F9 0000 0005          LEA    ARRAYB,A1   ;A1 POINTS TO ARRAY_B
00000016  103C 0005                MOVE.B #5,D0        ;FIVE ELEMENTS TO BE
                                                      ; COPIED
0000001A  12D8            LOOP:   MOVE.B (A0)+,(A1)+ ;COPY ELEMENT OF A TO
0000001C  5300                    SUB.B  #1,D0        ;ELEMENT OF B
0000001E  0C00 0000               CMP.B  #0,D0        ;FIVE OF THEM COPIED ?..
00000022  66F6                    BNE.S  LOOP         ;NO.. COPY ONE MORE.
00000024                  DONE:


         Lines Assembled :  23             Assembly Errors :  0
```

B. Sample Program #2

Source Code

```
;
;       /** Sample2.asm **/
;
;       The following program is to give an idea about the usage
;       of Coprocessor commands. Here the instructions are given
;       in their open form, the mnemonics of these commands are
;       given in the comment field.
;
;
        DC.W    $F200   ;FSINCOS.X FP4,FP5,FP6
        DC.W    $12B6   ;FP4 <- X (Prior to execution)
                        ;FP5 <- SINE(X),
                        ;FP6 <- COSINE(X)

        DC.W    $F23C   ;FMOVE.L #6,FP6
        DC.W    $4300   ;
        DC.W    $0000   ;
        DC.W    $0006   ;

        MOVE.L  #2,D6   ;
        DC.W    $F206   ;FADD.L D6,FP6
        DC.W    $4322   ;
```

LIST OF REFERENCES

1. Motorola, MC68000 Educational Computer Board User's Manual, 1982.

2. Motorola, MC68020 32-Bit Microprocessor User's Manual, Prentice-Hall, 1985.

3. Motorola, MC68881 Floating Point Coprocessor User's Manual, 1985.

4. G. J. Lipovski, "Communication Systems," in 16- and 32-Bit Microcomputer Interfacing, Prentice-Hall, preprint.

5. Tugcu, Y., "Design and Implementation of a MC68020 Based Educational Computer Board", Master's Thesis, Naval Postgraduate School, Monterey, Ca., Dec 1989.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Chairman Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
4.	Prof. Gerald J. Lipovski Department of Electrical and Computer Engineering The University of Texas Austin, Texas 78712	1
5.	Prof. Jon T. Butler, Code 62BU Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
6.	Prof. C. Yang, Code 62YA Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
7.	Prof. Frederic Terman, Code 62TZ Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
8.	Deniz Kuvvetleri Komutanligi Personel Daire Baskanligi Bakanliklar - Ankara / TURKEY	1
9.	Deniz Harp Okulu Komutanligi Tuzla - Istanbul / TURKEY	1

10. Yavuz TUGCU 1
Asagi Eglence Mercimek Sokak
41/20 Etlik - Ankara / TURKEY

11. Mustafa Yavuz UZUNSOKAKLI 1
Sancaktepe Mahallesi
4/5 Sokak, No:1 Daire:4
Bagcilar - Bakirkoy
Istanbul / TURKEY

Thesis
U96
c.1

Uzunsokakli
Design and implementation of a debugger for MC68020 based Educational Computer Board.

Thesis
U96
c.1

Uzunsokakli
Design and implementation of a debugger for MC68020 based Educational Computer Board.

AMERICAN BINDERY
TOPEKA KANSAS

thesU96
Design and impementation of a debugger f
3 2768 000 87951 4
DUDLEY KNOX LIBRARY